ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Steve Maassen

Supervised Learning of the Safe set in Autonomous Driving

Semester Thesis

Automatic Control Laboratory
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Alexander Liniger
Prof. Dr. John Lygeros

June 2017

Contents

Abstract

Guaranteeing safety in autonomous driving applications is a challenging task, due to the complexity of the models and the road constraints. Recent work showed that for autonomous racing where the road/track constraints are known a-priori the problem can be tackled by using a simplified model and viability theory. For this case the viability kernel describes all safe states. However, if the road constraints are not known a-priori this method can not be used due to the long computation times of the viability kernel algorithm. In this thesis we study the use of supervised learning to predict the viability kernel for an arbitrary curve. Therefore, a support vector machine (SVM) is trained with a library of precomputed turns of different radii and opening angles. We demonstrate this approach by considering a library of curves motivated by the ORCA project and show that it is indeed possible to learn a general curve using a SVM. Finally, it is shown that the trained SVM classifier is able to predict the viability kernel of yet unseen curves.

Nomenclature

Symbols

X	Absolute X coordinate	[m]
Y	Absolute Y coordinate	[m/s]
φ	Absolute orientation	[rad]
v_x	Longitudinal velocity	[m/s]
v_y	Lateral velocity	[m/s]
ω	Yaw rate	[rad/s]
δ	Steering angle	[rad]
$F_{i,j}$	Force at tire i and in direction j	[N]
m	mass of the car	[kg]
I_z	moment of inertia in z	[kg $\cdot$ m^2]
l_f	distance from CoG to front tire	[m]
l_r	distance from CoG to rear tire	[m]
$\bar{v}_i$	stationary velocity	[m/s]
R	curve radius	[m]
α	curve opening angle	[m]
w_t	stationary velocity	[m]
K^j	j^{th} iteration of the viability kernel	
l_{aug}	augmentation length	[m]
l_{ext}	augmentation length	[m]
φ_w	final orientation window size	[rad]
φ_d	curve direction angle at end or start	[rad]
d_{grid}	grid resolution	[m]
x^i	feature vector of sample i	
x^i_j	j^{th} entry of the feature vector of sample i	
y^i	label of sample i	
w	SVM weight vector	
ξ_i	slack variable of contraint i	
N_s	number of training samples	
d_i	dimension i	
$\phi(x)$	SVM kernel function	
γ	SVM kernel parameter	

C	C-SVM cost parameter

Indicies

x	Longitudinal direction
y	Lateral direction
f	Front wheel
r	Rear wheel
k	Time step k

Acronyms and Abbreviations

SVM	Support Vector Machine
ANN	Artificial Neural Network
MPC	Model Predictive Controller
PWM	Pulse Width Modulation
QP	Quadratic Program
ORCA	Optimal RC Autonomous Racing
CoG	Center of Gravity
ETH	Eidgenössische Technische Hochschule
RBF	Radial Basis Function

Chapter 1

Introduction

The research area of Control Design for autonomous driving has seen a lot of attention in the last few years. Companies like Tesla or Goggle are pushing towards a driving situation where more and more duties and responsibilities are given to the car itself. Obviously, safety plays a major role and is one of the key reasons why autonomous driving is not fully established today, especially since it is hard to guarantee safety at all times. This could be established by guaranteeing that the autonomous car always applies safe inputs which would guarantee that it never violates constraints, like being on the track. However, this is a difficult problem that has not yet been solved for an arbitrary road system and environments that are changing continuously. When considering autonomous racing where the car's operating point is on the edge of tire friction the problem becomes harder due to non-linearities but at the same time, the track is often known in advance so it is possible to tackle the safety problem.

One approach to establish safety is viability theory, where for given dynamics ,and input and state constraints, viability theory allows to calculate a viable set in the state space where every point is guaranteed to have an input that drives the system to another safe state. Hence by keeping the dynamics in the so called viability kernel, safety can be guaranteed. The main issue with the viability kernel is that, in most cases, it can not be calculated analytically. This is why the viability kernel is often calculated by discretizing the state space, and by using the viability kernel algorithm. The detailed functioning of the algorithm and the adaptations made for this problem is described in Section 2.1 . The main problem here is that, due to the gridding, the algorithm suffers from the curse of dimensionality which limits the number of dimensions of the system and the discretization exactness.

1.1 Goals

The goal of this semester project is to investigate if the viability kernel can be approximated with the help of supervised learning. By calculating the viability kernel for a library of curves with the help of an adapted viability algorithm we try to create a data set that can be used to train a classifier. This should allow to predict the viability kernel for an arbitrary curve or a sequence of road

elements and no longer only for a known track like in autonomous. At the same time computation time for evaluation could be drastically improved since all the training could be done in advance. Concretely this thesis aims to make a Proof of Concept of this principle for the autonomous RC Racing ORCA project [1] established by the automatic control lab (Ifa) at ETH Zurich.

1.2 Related Work

Some research has been done where supervised learning was used to either calculate or learn the viability kernel for specific situations. In [7] a support vector Machine (SVM) was used to adapt the classical viability kernel algorithm described in Section 2.1. After every iteration step, they approximated the discrete viability kernel with a continuous SVM. In the next step, while evaluating the difference inclusion, instead of checking an intersection with the discrete viability kernel iterate, the before trained SVM iterate is used. This was applied to a bike riding problem in [6] where they included a multi resolution grid to their method.

In [11], viability theory in combination with SVM's was used to investigate active safety during cornering maneuvers. The viability kernel for one specific maneuver was calculated using the viability kernel algorithm and in a second step the kernel was approximated using an SVM classifier. However, this only concerned one specific situation and the SVM was merely used to obtain a continuous set rather than a discretized grid.

These works show, how SVMs can be used to get a more accurate calculation or a continuous version of the viability kernel but it differs from our approach which aims at classifying unknown curves and tracks. To the authors best knowledge, this has not attempted yet in the existing literature.

Chapter 2

Existing Work

2.1 Viability Theory

2.1.1 Main Idea

Viability theory is a field of mathematics that is often used when dealing with dynamical systems with state and input constraints. It allows to prove safety for a given system and a given initial condition. At the core of viability theory stands the viability kernel which describes the subset of all states that can be considered as "safe" or viable. If at a certain point in time your system is in this set, there exists an input that can drive it to another point in the viability kernel. We will use this Section to briefly describe the mathematical concepts of viability theory and the resulting viability kernel algorithm.

2.1.2 Discrete Viability Theory

Discrete viability theory addresses the question: for which initial conditions does there exist a solution to a difference inclusion, which stays within a constraint set forever [3]? For the following, consider a controlled discrete-time system $x_{k+1} = f(x_k, u_k)$, where $x \in \mathbb{R}^n$ is the state, $u \in U \subset \mathbb{R}^m$ is a control input and $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ is a continuous function describing the dynamics. One can write this system as a difference inclusion as the following

$$x_{k+1} \in F(x_k), \qquad \text{with } F(x) = \{f(x, u) \mid u \in U\} \tag{2.1}$$

2.1.3 Viability Kernel Algorithm

Definition 1. *A set $D \in \mathbb{R}^n$ is a **discrete viability domain** of F if $F(x) \cap D \neq \emptyset$ for all $x \in D$. The **discrete viability kernel** of a set $K \subset \mathbb{R}^n$ under F, denoted by $Viab_F(K)$, is the largest closed discrete viability domain contained in K. [2]*

The viability kernel can be calculated with the following viability kernel algo-

rithm

$$K^0 = K,$$
$$K^{n+1} = \{x \in K^n \mid F(x) \cap K^n \neq \emptyset\} \tag{2.2}$$

Then the discrete viability kernel is defined by the following proposition

Proposition 1. *[2] Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be a upper-semicontinuous set-valued map with closed values and let K be a compact subset of Dom(F)*

$$Viab_F(K) = \bigcap_{i=0}^{n} K^n \tag{2.3}$$

Since the algorithm requires operations on arbitrary sets which is not implementable, one has to use a discretized grid instead of a continuous state space. In order to get an approximation of the viability kernel in the case of discretization, one has to consider expansion of the resulting set valued-map [2]. For simplicity, details are omitted here and can be read in [2] which further deals about inner approximations.

2.2 Autonomous RC Racing

This thesis originates from the ORCA (Optimal RC Racing) [1] project at ETH, which consists of developing a testbed for autonomous racing. Although this thesis will not include experimental tests on the before mentioned project, it clearly aims to show that a real-time application could be possible and could be tested on this setup. The project consists of 1:43 dnano RC cars racing autonomously around a 3x3 meter track (fig 2.1). A vision system serves as sensor to estimate the state of the cars, which gets distributed to standalone controller PCs, which then calculate an optimal input. This is then send to the cars via bluetooth, which execute the input. In the following sections the research that has been done in [8] is presented, since it is needed for this thesis.

Figure 2.1: Track and Kyosho dnano cars used in the experimental setup [8]

2.2.1 Car Model

For the model of the RC car, a nonlinear bicycle model (Figure 2.2), using the Pacejka tire model [4], is chosen. This model is suited for the application of autonomous racing, since it captures important dynamics such as saturation of the nonlinear tire force. This is essential, since, other than in normal autonomous driving where speeds are usually low, in the case of autonomous racing speeds and tire forces are as high as possible. This model is also suited for the case where the tire force exceeds the friction force (drifting). The states X, Y, and

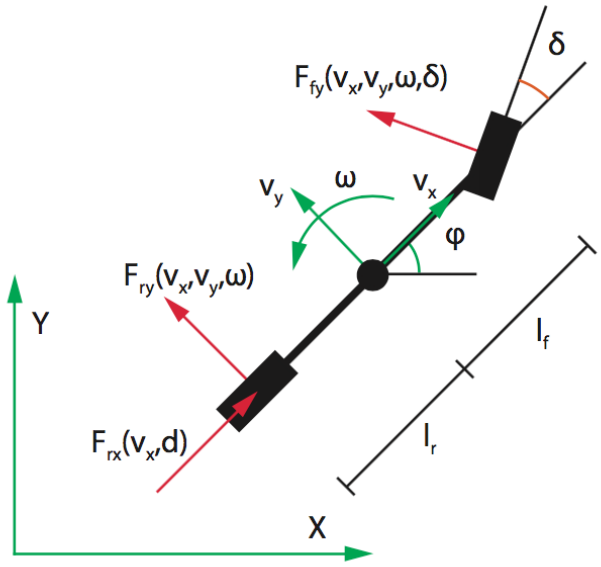

Figure 2.2: Bicycle Model [8]

φ describe the absolute position of the car measured in the inertial coordinate frame. Further, v_x, v_y and ω correspond to the forward velocity, lateral velocity and angular velocity respectively and are measured in the local coordinate frame attached to the car at the center of gravity (CoG). The steering angle δ and a pulse width modulation (PWM) act as inputs to the system. The complete equations of motion can be seen in (2.4).

$$
\begin{aligned}
\dot{X} &= v_x \cos(\varphi) - v_y \sin(\varphi), \\
\dot{Y} &= v_x \sin(\varphi) - v_y \cos(\varphi), \\
\dot{\varphi} &= \omega, \\
\dot{v}_x &= \frac{1}{m}\Big(F_{r,x}(v_x, d) - F_{f,y}(v_x, v_y, \omega, \delta)\sin(\delta) + m v_y \omega\Big), \\
\dot{v}_y &= \frac{1}{m}\Big(F_{r,y}(v_x, v_y, \omega) - F_{f,y}(v_x, v_y, \omega, \delta)\cos(\delta) + m v_x \omega\Big), \\
\dot{\omega} &= \frac{1}{I_z}\Big(F_{f,y}(v_x, v_y, \omega, \delta) l_f cos(\delta) - F_{r,y}(v_x, v_y, \omega)\Big) l_r,
\end{aligned}
\tag{2.4}
$$

where $F_{r,x}(v_x, d)$ is the force of the drive train, $F_{r,y}(v_x, v_y, \omega)$ and $F_{f,y}(v_x, v_y, \omega, \delta)$ are the lateral forces at the tires given by the Pacejka tire model [4] depicted in equations (2.5).

$$\begin{aligned} &F_{r,x}(v_x, d) = (C_{m1} - C_{m2} v_x) d - C_r - C_d v_x^2, \\ &F_{r,y}(v_x, v_y, \omega) = D_r \sin(C_r \arctan(B_r, \alpha_r)), \\ &F_{f,y}(v_x, v_y, \omega, \delta) = D_f \sin(C_f \arctan(B_f, \alpha_f)), \\ &\text{with } \alpha_r = \arctan\left(\frac{\dot{\varphi} l_r - v_y}{v_x}\right) \quad \text{and} \quad \alpha_f = \arctan\left(\frac{\dot{\varphi} l_f - v_y}{v_x}\right) + \delta. \end{aligned} \tag{2.5}$$

Here, B, C, and D are parameters used to tune a semi-empirical curve. Further, m is the mass of the car, l_r and l_f the distances to the rear and front tire respectively, and I_z is the moment of inertia around the Z axis.

2.2.2 Constant velocity trims

In order to simplify the nonlinear dynamics, the differential equations describing the velocities are simplified by assuming stationary velocities. Therefore a grid of modes (e.g. triplet of constants $\bar{v}_x$, $\bar{v}_y$, and $\bar{\omega}$) is used to create a hybrid like system. These stationary velocities can be found by setting the accelerations to zero and picking different pairs of forward velocities v_x and steering angles δ. A fact to consider here is that this includes modes where the relation v_y/v_x exceeds the no slip constraint resulting in "drifting" modes. By using this simplification the still nonlinear kinematic part of the the model (e.g. equations 1-3 in (2.4)) can be solved analytically. When defining a time T_{pp} for which the analytical solution is integrated we get the so called trims which can be seen as steady state arc trajectories. The difference equation needed for the viability algorithm can then be obtained as follows

$$\begin{aligned} x_{k+1} &= x_k + \frac{\bar{v}_x}{\bar{\omega}}\big(\sin(\bar{\omega} T_{pp} + \varphi[0]) - \sin(\varphi[0])\big) \\ &\qquad + \frac{\bar{v}_y}{\bar{\omega}}\big(\cos(\bar{\omega} T_{pp} + \varphi[0]) - \cos(\varphi[0])\big), \\ y_{k+1} &= y_k + \frac{\bar{v}_y}{\bar{\omega}}\big(\sin(\bar{\omega} T_{pp} + \varphi[0]) - \sin(\varphi[0])\big) \\ &\qquad - \frac{\bar{v}_x}{\bar{\omega}}\big(\cos(\bar{\omega} T_{pp} + \varphi[0]) - \cos(\varphi[0])\big), \\ \varphi_{k+1} &= \varphi_k + \omega T_{pp}, \\ m_{k+1} &= u_k \qquad\qquad u_k \in U(m_k), \end{aligned} \tag{2.6}$$

where $\bar{v}_x(m), \bar{v}_y(m)$ and $\bar{\omega}(m)$ are defined by the mode m which is the current velocity state of the system. The input of the system then boils down to choosing the mode for the next time step from $U(m_k)$. The latter input set consists of up to 27 modes that are near to m_k (e.g. that can be reached quickly enough from m_k). This resulting state automata is illustrated in Figure 2.3 where the current mode is 4, and the arrows indicate the possible choices for the next mode.
At the end, for every current mode, we get a collection of input trims which is used by the controller as described in Section2.2.3. Figure 2.4 shows the car

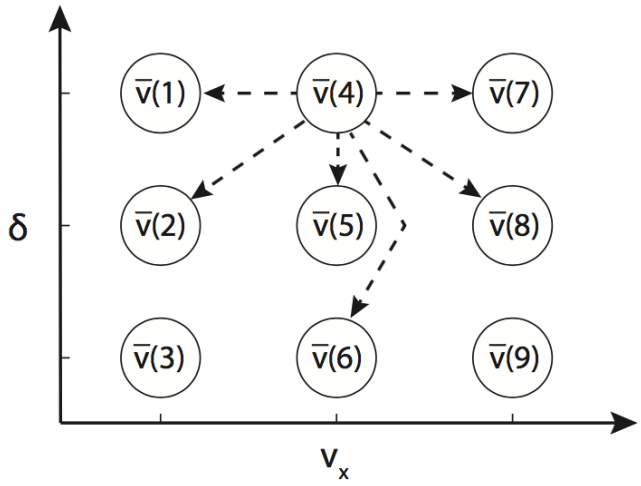

Figure 2.3: Schematic drawing of allowed transitions from $\bar{v}(4)$ to other velcoity points [9].

with $\varphi = \pi/2$ on a straight piece of track and the trajectories resulting from the modes $m = 28 - 38$.

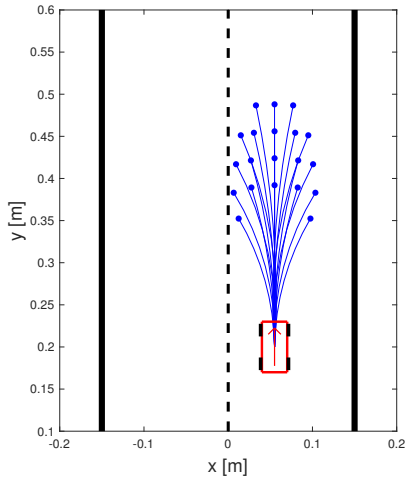

Figure 2.4: Car on track, plotted with a selection of trims

2.2.3 Control Scheme

In previous work, a two level hierarchical controller was designed and implemented, which consists of a high level path planner and a low level reference tracking model predictive control (MPC) [8]. The hight level path planner generates all allowed trajectories of the before mentioned model, and discards all

trajectories which leave the constraints. Of all the feasible trajectories the one with the largest progress is selected. 2.3).
In order to measure the progress of planned trajectory, the endpoint of each trim is projected onto the middle line. In the implementation of [8], the path planner could use two consecutive trims to construct an optimal trajectory. This results in a more foreseeing trajectory but is obviously computationally more expensive since the computation time grows exponentially with the number of consecutive trims. As before mentioned, the path planner can also initiate drifts when being in a mode that is "near" a drift mode and allows a transition into it.

The low level controller then gets the optimal trajectory from the path planer and solves an MPC problem in order to track the chosen trim as accurate as possible. In order to solve the nonlinear control problem, local convex approximations are used to build QPs.

2.2.4 Viability Kernel for the Whole Track

In previous research, the concept of the viability theory has been applied to the ORCA project [10]. The viability kernel algorithm has been used to calculate the kernel for the whole 3x3m track which took up to 397.674 seconds depending on the grid resolution an the exact algorithm used. The kernel was included in the higher level controller, by not only discarding all trajectories which leave the track but also the ones which leave the viability kernel. This reduced online computation time. Results showed that due to the reduced computation time, the prediction horizon could be extended and the controller generally was more foreseeing and produced better trajectories.

Chapter 3

Methods

3.1 Adapted Viability Kernel Algorithm

In order to generate a library of curves as training data for the supervised learning, the discrete viability algorithm from 2.1.3 has to be adapted. Firstly, the standard algorithm needs a closed set as for example a closed circuit, a single curve, however, has a beginning and an end, which are both open. Further, everything that comes after and before the turn is uncertain and undefined and should ideally not contribute or influence the learning data. In Section 3.1.2 we will present a solution to this problem and an adaptation of the algorithm. In Section 3.1.3 we will come up with a method on how to handle the uncertainty on the boundaries of the curve, meaning the start and the end.

3.1.1 Curve Definition

To begin with, we define a curve by it's curve radius and opening angle. The cartesian coordinate system has it's origin in the middle of the turn at the starting point. For convenience we assume in the following that the turn always starts by going in positiv y-direction and that the middle line is tangential to the y-vector (Figure 3.1 left). Since the viability kernel algorithm checks every angle φ we do not attach an intrinsic direction to the curve. Therefore we only consider curves going to the right. A left turn can then be seen as a right turn traveled through in reverse. Note, that a polar coordinate system with the origin in the center of the turn seems like another valid choice, especially when thinking about the later feature definition. However, in several tests we saw that the choice of a polar coordinate system deteriorates the smoothness and continuity of the learned viability kernel, which is why we rejected it. The curve width w_t is chosen so that it matches the track width of the ORCA project (e.g. 30 cm).

3.1.2 Domain Extension

With the curve definition in Section 3.1.1 we encounter one major problem when trying to feed it to the viability kernel algorithm. Let's assume we would define the initial set K^0 as the raw curve (e.g. green region in Figure. 3.1 on the right). The algorithm would see traversing the start or end line of the curve as

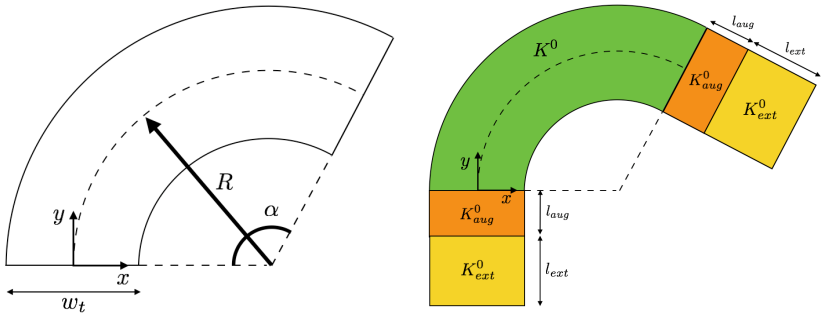

Figure 3.1: left: curve definitions; right: curve extetntion

constraint violation. Therefore, all states that have inputs, which would allow the car to make the turn but exit it at the end, would be marked as unsafe. In other words, the car would be forced to stop before exiting the curve in order to be safe. Since this is clearly not what we would like to learn with the machine learning tool, we came up with an extension of the curve in both, start and end directions. Ideally, the solution would be to extend the curve ends into infinity, which would result in a unbounded set not allowed in theory (and practice). However, choosing a finite but long enough extension is sufficient since the car can stop in a finite distance. Therefore when the total extension length (eg. $l_{ext} + l_{aug}$ in Figure 3.1 on the right) is chosen large enough we should get the same result as in the case of infinite extension.

Secondly, even with a finite extension, the number of points that would be obtained by griding the curve in x and y is still unreasonably big. Further, the part of the viability kernel on the straight extension would carry little, to no, information that would be learned. Therefore we split the extension into two regions defined by l_{aug} and l_{ext}. In the the augmented region K^0_{aug} (orange region in Figure 3.1 on the right) the viability algroithm is still evaluated. In the extended region K^0_{aug} (yellow region in Figure 3.1 on the right), merely constraints are enforced. This allows us to reduce the number of points fed to the algorithm to a reasonable number of roughly 200 on average with a 0.04 m grid resolution. In K^0_{aug}, not only the track constraint is verified but also the resulting orientation of the car is constrained. This is done by defining an angle window with an opening of φ_w around the angle of direction φ_d of the curve resulting in $[\varphi_d - \varphi_w, \varphi_d - \varphi_w]$. If the car reaches the extension region from a certain state with a certain input, the orientation of the car at the end of the maneuver has to lie in this angle window in order for the input to be marked as feasible. This rejects situations where the car would be unable to recover in a future step.

3.1.3 Boundary Uncertainty

The main goal of this thesis is to find a way to calculate the viability kernel algorithm for an arbitrary sequence of curves. This means that the viability kernel for one curve should be learned without any assumptions on the next part of the track. This is not an easy task since the kernel of the first turn

is clearly influenced when another curve follows directly. In order to somehow handle this uncertainty, we adapted the curve extension from 3.1.2 by tightening the track directly after the end of the curve. This is done with the help of an error function also known as Gauss error function

$$\text{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt \tag{3.1}$$

As seen in Figure 3.2, the end of the inner and outer borders of the curve is patched with the error function and then extended linearly. This way we keep the car "safer" by forcing trajectories into the middle of the track in order to be categorized as safe. Consequently the car would have more room to react to sudden changes like an immediate next turn.

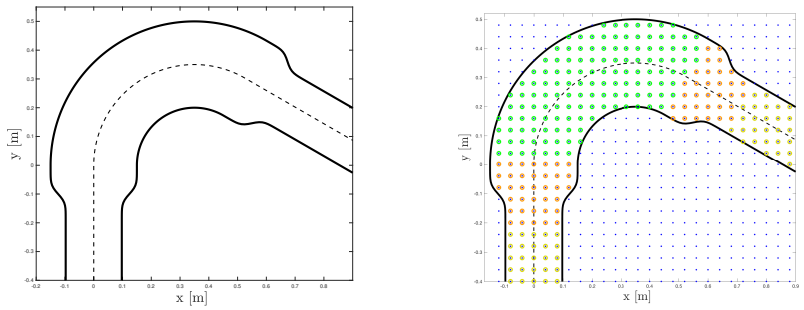

Figure 3.2: left: curve with error fucntion; right: curve with grid

We are then left with the final definition of a curve which is gridded in x and y with a grid resolution d_{grid} which is shown in Figure 3.2 on the right for $R = 0.35$ m, $\alpha = 2\pi/3$, $l_{aug} = 0.2$ m, $l_{ext} = 0.6$ m, and $d_{grid} = 0.04m$. For the error function parameters, we used a narrowing of 35 % over a tangential distance of 0.2 m. All this parameters are further investigated in Section 3.3.1..

3.1.4 Resulting Algorithm

Algorithm 1 describes how the viability kernel is calculated with our definition. One remark that could be made, is that, in the real algorithm, not only the endpoint of the different trims is checked but it is guaranteed that the trims do not exit the track in the middle while re-entering at the end. The while loop usually terminates in 10 to 15 iterations for the curves we used and returns the training data for the supervised learning. A selection of calculated kernels can be seen in Figure 3.3.

3.2 Choice of machine learning tools

Machine learning is a trending topic for several different applications, ranging from life sciences over economics to engineering. The main idea is to use a large training set to learn a so called classifier, which, after training, can predict a label when presented with a new sample that is not included in the training set.

Algorithm 1: Adapted Viability Kernel Algorithm

```
Data: K^0_algo = K^0 + K^0_aug , K^0_ext
Result: K^∞_algo
begin
    n ⟵ 1
    K^n_algo ⟵ ∅
    while K^n_algo ≠ K^{n-1}_algo do
        for every point p ∈ K^{n-1}_algo do
            for every possible input u do
                endpoint=DifferenceEquation(p,u)
                if endpoint ∈ K^0_algo then
                    if endpoint ∈ K^{n-1}_algo then
                        K^n_algo ⟵ K^n_algo ∪ p
                else if endpoint ∈ K^0_ext then
                    if orientatiion ∈ angleWindow then
                        K^n_algo ⟵ K^n_algo ∪ p
        n ⟵ n + 1
        K^n_algo ⟵ ∅
    K^∞_algo ⟵ K^{n-1}_algo
```

This is done by estimating relationships between variables, often called features, which is commonly known as regression. There are several approaches on how to create such a classifier or regressor. The most recent success has been made with Neural Networks and Support Vector Machines which are briefly explained in the following Sections.

3.2.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a classifier whose methodology is based on the functioning of neurons in a biological brain. The main idea is that a selection of inputs are used as triggers to a first layer of neurons. Every neuron or a combination of several neurons then serves as an input for a number of second layer neurons. A neuron only fires when it's input value is high enough and the output can be weighted by adjustable weights w_{ij}. Figure 3.4 illustrates this principle and shows how the output is dependent of some combination of the inputs and the weights. The training samples for which the outputs are known is then used to determine the weights in an optimal manner so that as few as possible samples are misclassified. In our case, the training set is the set of grid points of the curve and the output is either safe (1) or unsafe (0). We tried to implement an ANN for our problem but found out that a Support Vector Machine resulted in less training time and comparable or better results which is why we will not go more into detail on this topic.

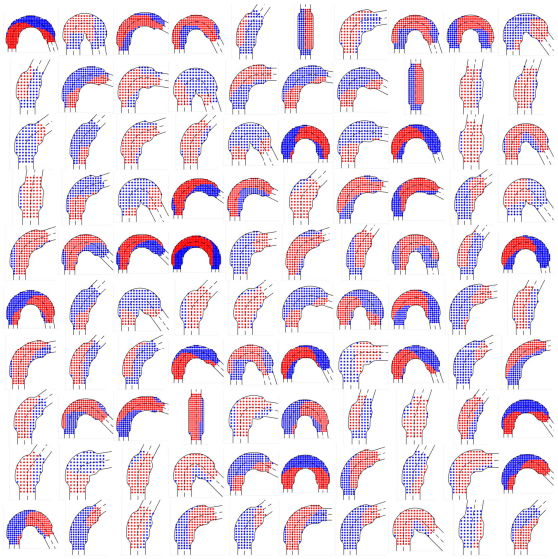

Figure 3.3: a selection of different calculated viability kernels for given curve, mode and orientation. Red are safe points, blue are unsafe points

3.2.2 Support Vector machines

Support Vector Machines (SVM) have seen a lot of success in recent years and have become one of the favorite tools to learn classifiers. The main idea is that you want to divide a set of training points into one (in case of single class classification) or multiple regions. This is done by creating a separating hyperplane that is determined by an optimization problem.

3.2.3 Main Concepts of C-SVM

Let's define a sample i as (x^i, y^i) where $x^i \in \mathbb{R}^n$ is the feature vector with n features and y^i is the label. As features, we understand values assigned to the sample that have an influence on the outcome of y. The goal of the training is to find a hyperplane defined by $f(x) = 0$ where

$$f(x^i) = \langle w, x^i \rangle + b \tag{3.2}$$

that separtes the data set into two classes. $\langle \cdot, \cdot \rangle$ denotes the scalar product of two vectors, $w \in \mathbb{R}^n$ is a weight vector, and b is a constant offset. $f(x)$ takes on positive values if the predicted label is 1 and negative values when the predicted value is -1. Let's consider a n-dimensional feature space, with samples i $x^i = [x_1^i, ..., x_n^i]$ and two classes (e.g. positives and negatives). As seen in Figure 3.6, illustrated for $n = 2$, we try to find the function $f(x)$, in this case representing a line $(w_1 x_1 + w_2 x_2 + b = 0)$, which separates the two regions by maximizing the margin $2/\|w\|$ to the nearest points. The SVM problem can

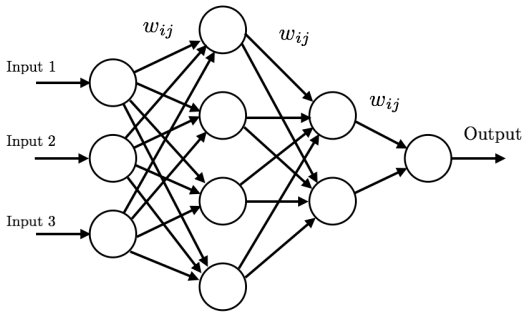

Figure 3.4: illustration of an artificial neural network

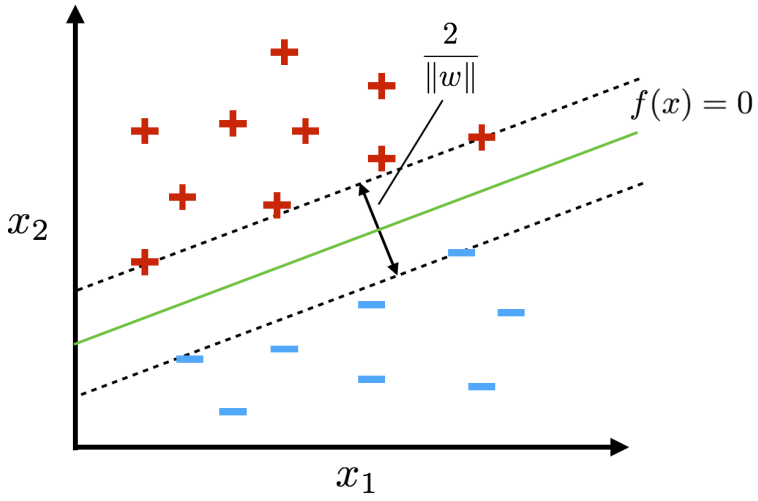

Figure 3.5: illustration of the SVM concept

then be solved by solving the minimization problem

$$\begin{aligned} & \min \ \frac{1}{2} w^T w + C \sum_{i=1}^{N_s} \xi_i \\ \text{s.t.} \quad & y^i f(x^i) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, ..., N_s \end{aligned} \tag{3.3}$$

where ξ_i is a slack variable associated to constraint or sample i, and N_s is the total number of samples. C is a tuning parameter that can be used to punish outliers. In other words, by increasing C a hard constraint violation (or an outlier) adds more cost to the cost function. The minimization variable is the weight vector w and the constant b. In the following, every time we refer to SVM, we mean C-SVM.

Stochastic Gradient Descent SGD

The C-SVM method performs well when the number of samples is not too big (e.g. $\leq$ 1'000'000). If the sample size is larger, other optimization methods are often used. One of these is Stochastic Gradient Descent. Instead of calculating the true gradient of the optimization function, SGD approximates it by only looking at one sample at the time. Since we have a large number of samples (e.g. up to 500'000'000 points) we looked into the benefit of this method. However, due to the high density of information in our specific problem, meaning we have no outliers and every point on the edge of the safe region is needed, SGD did not perform well. Hence further mathematical details on this method is omitted.

3.2.4 Normalization

Something that needs to be done almost every time when handling a classification problem is normalization of the data. This is done in order to prevent numerical issues as well as unbalanced sensitivity in different feature dimensions. We applied hard normalization to all our feature dimensions where, first the maximum and minimum of each feature dimension in the training data set is determined, this values are then used to scale all training data points in between -1 and 1. For predicting a new sample, it's features are normalized with the help of the transformation used in the training phase.

3.2.5 The Kernel Trick

Pure SVM is a linear method that can only use linear manifolds to separate the data into two regions. Sometimes however, the data is not separable in this manner and so we can use the kernel trick, which allows us to handle nonlinear boundaries. the main idea is to expand the feature space in a higher dimension in order to separate the data linearly. In Figure 3.6, this concept is illustrated by taking a 2D feature space and expanding into a 3D feature space. This way the two classes can be separated and a nonlinear boundary in the actual feature space is possible.

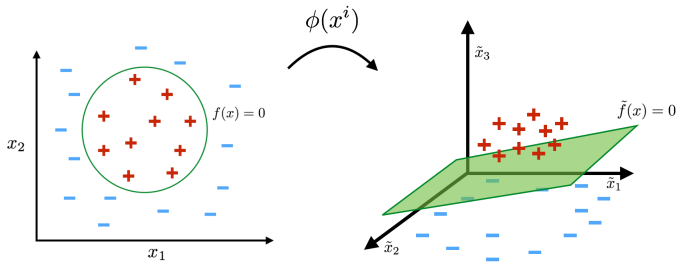

Figure 3.6: illustration of the Kernel methods for a 2D feature space

This transformation is done by using a so called kernel function

$$\phi : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, x \to \phi(x) \tag{3.4}$$

where as before mentioned $d_1 < d_2$. In this thesis, we used a Radial Basis Function (RBF) also called a gaussian kernel (see eq. (3.5)), which is commonly used and usually leads to good results for a large set of problems. (x^i, x^j) is a pair of features and γ is a tuning parameter that influences the "smoothness" (e.g. the maximum curvature).

$$\phi(x^i, x^j) = \exp(-\gamma \|x^i - x^j\|^2) \tag{3.5}$$

3.2.6 Kernel Approximation

One of the main drawbacks of using kernel methods is that it does not scale very well with large data sets, since the feature space grows with the number of samples. Numerically spoken, for m samples with a feature dimension $d_1 = 1$, a $(m \times m)$ matrix has to be evaluated. When running into computational limits one can consider using a kernel approximation methods. This way, not every feature is used to construct the kernel matrix but a Monte Carlo sampling is used to determine a fixed number of vectors. We tried this approach but were not successful with our data set. This is not surprise since, in the same way as with SDG, nearly all of our the training points are essential, and randomly choosing a small percentage of them to construct the kernel matrix leads to meaningless results.

3.3 Definition of the Curve Library

Every machine learning approach needs a set of data from which it can learn. The choice of this data set is crucial to the performance of the classifier which is why we discuss the different options for training sets and features in the following subsections.

3.3.1 Parameter Sensitivity

Firstly, we investigate the sensitivity of the parameters that we introduced in our curve the definition (see Section 3.1). We ran the adapted viability kernel algorithm several times while always changing only one parameter and plotted the results. This way we got a qualitative understanding on which parameters would be suited as features and which one could be set to constant values without changing the overall results. Naturally, the goal of this exercise was to find as few features as possible in order to improve training times, especially since we are using kernel methods. The reference values for the parameter used in the analysis are the following: $R = 0.35$ m, $\alpha = 2\pi/3$, $l_{aug} = 0.4$ m, l_{ext}=0.6 m, $\varphi = \pi/9$, and a reduction with erf(x) of 0.35%. Since, in a first step, we are mostly interested in a real time implementation working with the ORCA track we fix the track width to 0.3 m.

- **Curve Radius** R:

  By changing the radius of the curve from the smallest possible (e.g 0.15m) to larger values, we see that the safe regions of the viability kernel change. This was to be expected, since it is a parameter that has a big influence on the shape of the curve itself. Figure 3.7 shows the viability Kernel for three different radii and two different modes and orientations.

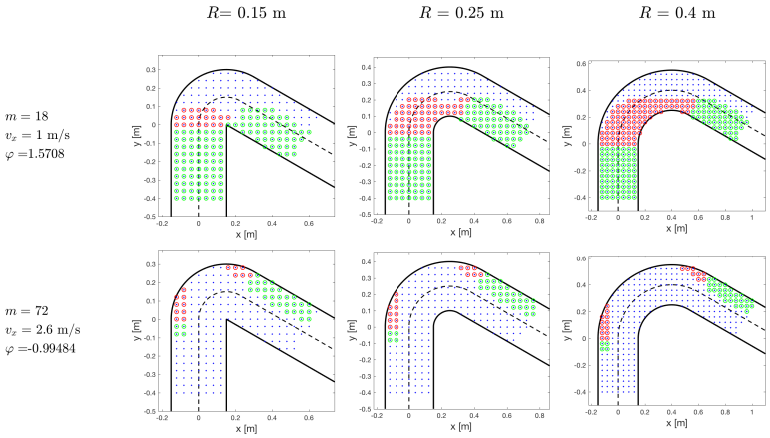

Figure 3.7: Viability Kernel for different radii and modes

In Figure 3.7, the red points represent all points belonging to the safe set in K^0 and the green points are safe point that are in K^0_{aug}. Since the latter set is merely a tool for handling the finite character of the problem, we should only consider changes of the red points as important. Even if the shape of the safe set changes, they do not change drastically, which leads us to believe that when including R as a feature, SVM can classify new curves in between of training curves. Another interesting result from the analysis is that the changes in higher modes (e.g. higher velocities) seem to be less significant.

- **Opening angle** α:

  It seems obvious that α is another parameter that should be included as a feature, since it also changes the shape of the turn dramatically. In Figure 3.8, three different curves for two different modes and orientations are shown.

  We see that for the small as well as for the fast velocity, the safe regions change and move. We also see that the set changes from one continuous set to a disconnected set.Therefore, α needs to be considered as a feature and it will be interesting to see how well the SVM handles these switches.

- **augmentation length** l_{aug}:

  The analysis of the changes for different l_{aug} is important since we want to chose it as small as possible in order to reduce computation time. The goal here, is therefore to find out from which augmentation lengths we do not see any changes anymore.

  In the same manner as before, Figure 3.9 shows the results from the

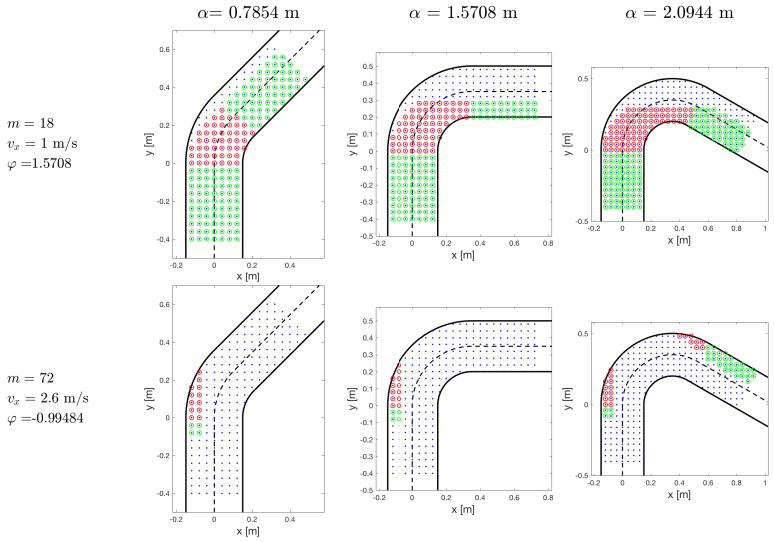

Figure 3.8: Viability Kernel for different α and modes

adapted viability kernel for different augmentation lengths and two different modes and angles. We see that for the $l_{aug} = 0.15m$ and the higher velocity, the result is clearly influenced by the shortage of the augmented region. This can mainly be said since there are no safe points in the augmented regions. In the case of $l_{aug} = 0.4m$ we see that the augmented region regions still has safe points which allows us to conclude that all influence by the finiteness of the problem is caught by this introduced boundary tool.

- **extension length** l_{ext}:

  The analysis of the influence of l_{ext} is merely discussed for the sake of completeness. It does not influence computation time and could be chosen as along as we want it to be. However, in Figure 3.10 we see that it has to be bigger than a certain length in order not to influence the shape of the viability kernel.

- **orientation window opening angle** φ_w:

  The angle window is also a measure that might influence the shape of viability kernel. However, it does not make any sense to include it as a feature since it is an artificial design parameter that has to be fixed. Figure 3.11 shows how the angle window changes the the results of the adapted viability kernel.

  Especially for the higher velocity and the smallest angle window we encountered a problem during testing. With the chosen orientations for the car, one might assume that the points in the augmented start Section

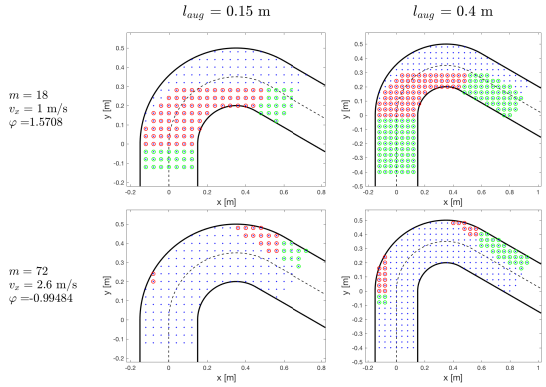

Figure 3.9: Viability Kernel for different l_{aug} and modes

should be considered as safe. Figure 3.12 shows the case of a window of 7.5° (orange region) and the two trims that are closest to the angle window.

The problem the algorithm encounters is that the trims merely miss the window in the first orientation shown (left side of Figure 3.12) . However, when changing the orientation just a little bit, nearly all points in negative y direction are safe again. This is due to the discrete character of the trim method and we get an alternation between safe and unsafe for a set of points when cycling through the orientation angle. This provokes bad results when trying to train an SVM due to the non-continuous changes in the safe set and does not represent the reality well. Therefore a larger angle window should be chosen to avoid such problems.

- **tightening function** erf(x):

  We only analyze the influence of the tightening of the curve by the error function and set the length over which the tightening happens equal to the augmentation length. The more we reduce the track width after the turn, the more we restrict the path planner and influence the trajectories. However, as discussed in Section 3.1.3, we want to somehow deal with the uncertainty after the turn. As a result there is no evident choice of the narrowing percentage and we merely apply an educated guess after analyzing the viability kernel for different values. Figure 3.12 shows the results for different reduction percentages. We can see that the safe set in K^0 is clearly influenced by the change of the parameter and that 50 % might be a bit too restrictive.

3.3.2 Feature Selection and Parameter Definition

With the analysis done in Section 3.3.1 we made the decision to only use R, and α as extra features for the SVM, especially when keeping in mind that a larger feature space has a severe influence on the training time. Hence the resulting training samples are given by

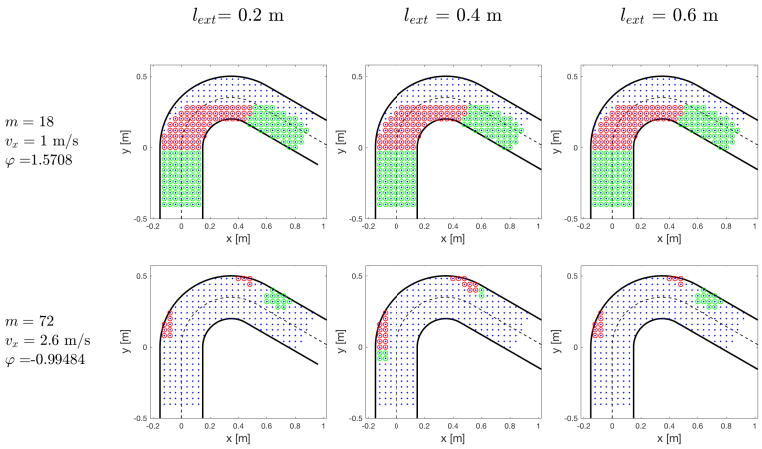

Figure 3.10: Viability Kernel for different l_{ext} and modes

$$x^i = [X^i, Y^i, \varphi^i, \bar{v}_x^i(m), \bar{v}_y^i(m), \bar{\omega}^i(m), R^i, \alpha^i] \quad y^i = \{1, 0\} \qquad i = 1, ..., N_s \tag{3.6}$$

We used $\bar{v}_x^i(m), \bar{v}_y^i(m)$, and $\bar{\omega}^i(m)$ for the velocity features instead of m since the latter has no physical meaning and is merely an index. By using physical parameters we can later use the SVM to also classify velocities that are not captured by the mode discretization. For l_{aug} a trade off value of 0.4 m was used and l_{ext} was conservatively chosen to be 0.6 m. ϕ_w was fixed to 20° in order to avoid above mentioned problems. Finally for the reduction of the error function we made an educated guess of 35 %.

3.3.3 Reducing the Number of Training Points

The safe regions we try to classify are obviously not linear, which is the reason why we have to apply a kernel method as described in Section 3.2.5. In order to be as efficient as possible in the training phase, we try to reduce the number of training points. Another motivation for reducing the points is the fact that, surely, not all the points carry information. Mainly, the points of interest are the points on the boundaries of the safe regions that will later on be chosen by the SVM as support vectors. We therefore applied an algorithm that reduces the the set of training points in X and Y and only chose every third angle in the orientation discretization to be included in the training set. This way, on average, the number of points per curve could be reduced to 60% of the original training set. The reduction algorithm takes as input a slice of the training set (e.g. X, Y-Grid for fixed angle and velocity) and outputs only the border

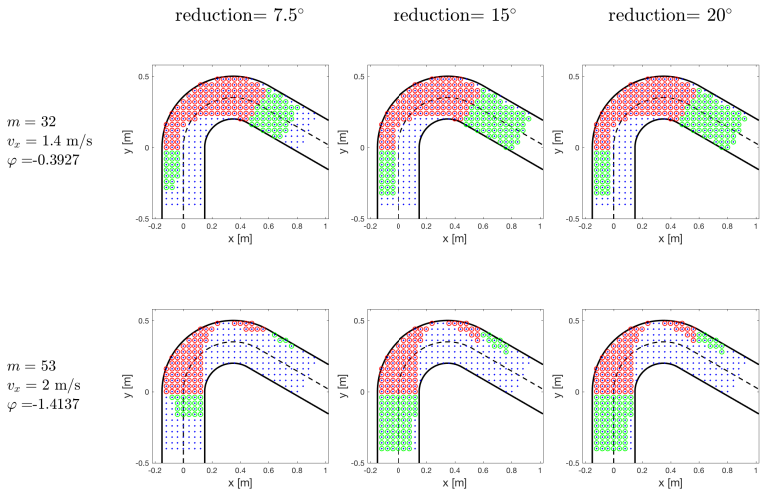

Figure 3.11: Viability Kernel for different φ_w and modes

points between the safe and unsafe regions as well as a coarser grid inside the continuous regions. An example can be seen in Figure 3.14 where the number of points for fixed orientation and velocity could be reduced from 205 to 143. As before, red encircled points represent safe points while blue ones represent unsafe points.

3.3.4 Implementation

The Curves and Grid where generated in MatLab. For the adapted viability kernel algorithm, the existing algorithm was edited and changed in Julia. Unlike MatLab, Julia is "just in time compiled" and offers greater speed when it comes to many iteration loops. For the training, predicting and handling of data we chose MatLab due to it's ease of use. Firstly we tried using only the build-in libraries for SVM classifiers. However we discovered that the computation time took way too long due to the interpreter based language. We therefore decided to use the precompiled library LIBSVM [5] written in C++ that comes with a MatLab interface. All analysis, the normalizing and the plotting where then again done in Matlab. Figure 3.15 shows a flow chart of the process.

3.3.5 Limitations

Despite all the efforts of keeping the feature space as small as possible and reducing the number of training points, we ran into feasibility issues concerning the training time. With the reduction methods applied to one single curve, e.g. only considering $(X, Y, \varphi, \bar{v}_x, \bar{v}_y, \bar{\omega})$, produced around 2.5 million points (depending on the curve parameters). Traditionally, C-SVM classifiers are suited well for

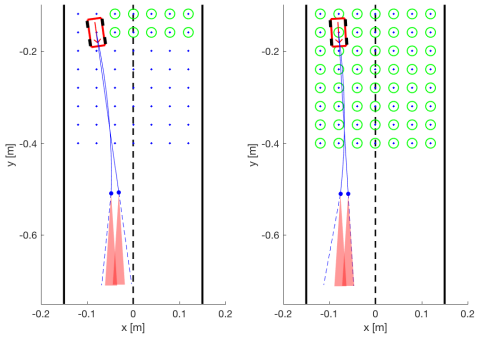

Figure 3.12: Illustration of the angle window principle

problems with 100.000 and less points which is substantially less then we have, so for one curve we are already at the boundary of the algorithm. In order to make the classification problem computationally feasible, we decided to drop the velocity features. The idea is to train 127 seperate SVM's which results in around 3 million training points for a complete library of curves. Note that this is not a problem in the real time phase since the path planner only considers the discrete velocity modes. With this trick, we not only reduced the number of points but also the feature space which is also beneficial for the computation time, when keeping in mind that the kernel method is used.

3.3.6 Real Time Implementation

Since the goal of the Thesis is to provide a proof of concept that a realtime implementation is possible when using supervised learning, we now discuss some further points that have to been taken into consideration. With the methods described in earlier Sections, the track constraints are not taken into considerations. Although only valid points on the track were taken into consideration, the obtained classifier could potentially classify a point off track as safe. Therefore, when the SVM is used by the path planner it has also to check the track constraint in order to chose a viable trajectory. This could not only be done by using methods used in [8], but another SVM could be trained that classifies a point as being on the track or not. In that case, the path planner would have to use both SVMs and combine them with a logical "and" in order to get the desired safe or unsafe classification. As proof of concept, we trained an SVM on the existing track that can be seen in Figure 3.16. As training points we created offsets of the track boundaries and gridded it with some resolution. As seen in the Figure, the SVM is able to approximate the real world track accurately. Concerning the computation time, both methods (e.g. SVM or track projection used in [8]) took rouhgly the same time on average.

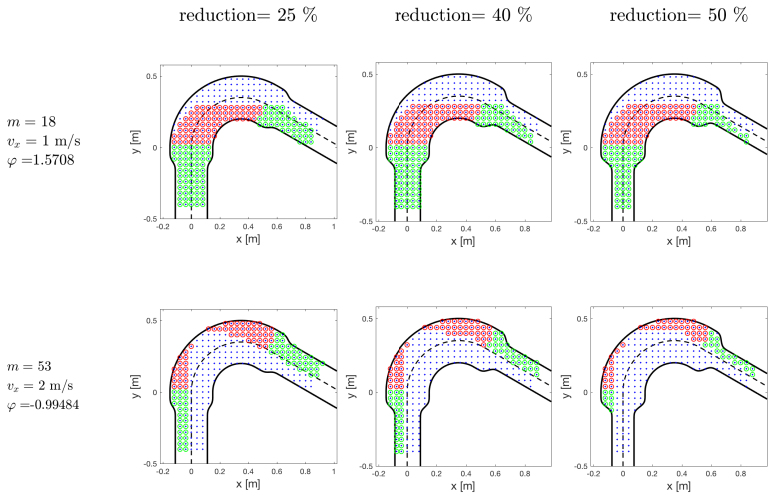

Figure 3.13: Viability Kernel for different reduction perc. and modes

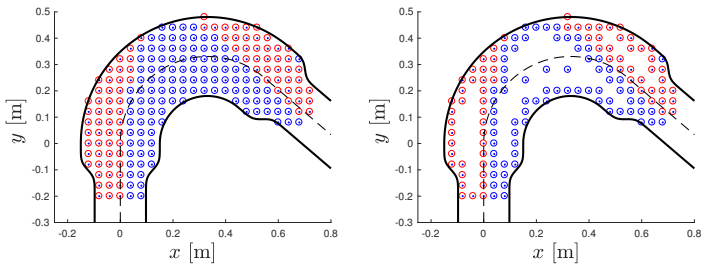

Figure 3.14: Left: Full set of train. points; Right: Reduced set of train. points

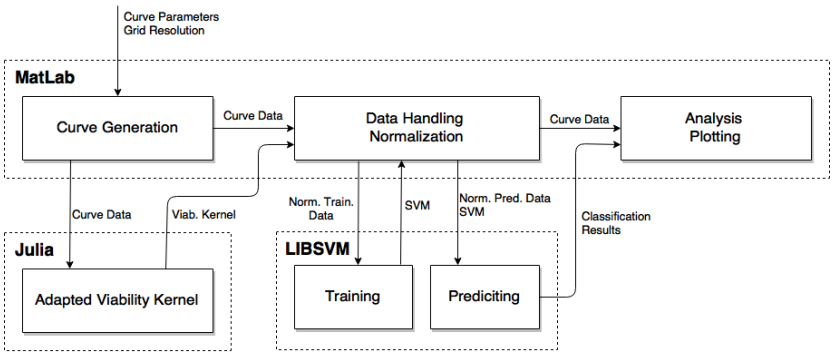

Figure 3.15: Left: Full set of train. points; Right: Reduced set of train. points

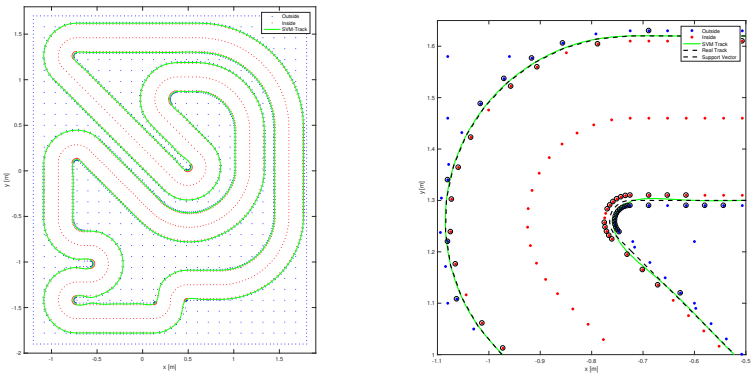

Figure 3.16: SVM Classifier for being on the track

Chapter 4

Results

4.1 SVM of one Curve

To begin with, we tried training one curve with an SVM that would then give us a continuous approximation of the viability kernel. We trained a curve with $R = 0.43m$ and an opening angle of $\alpha = 2.3562$. For training parameters we used $\gamma = 20$ and $C = 20$, which gave us satisfying results. The SVM was trained with 995'372 training points which was achieved by using the reducing algorithm from Section 3.3.3 and by taking every third angle resulting in 53 discrete orientations. The number of resulting support vectors was 99'957. The training took 287'512 seconds on a server with average machine specs. Figure 4.1 shows the prediction of the SVM for four different modes and orientations.

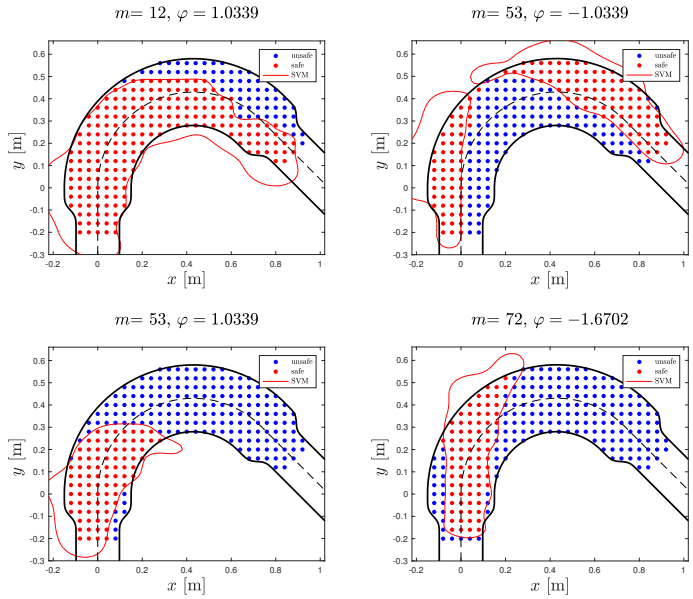

Figure 4.1: Different Predicition Results for One-Curve SVM

Note here that the top row plots show orientations which were used in the training set while the bottom plots show orientations that were not included. We see in both cases that the SVM does a good job in classifying the points rights even though eventual errors occur. As a quantitative measure we made a cross validation by looking at how many outliers we got in the training set and how many there are in the non-training set. Figure 4.2 shows the percentage of misclassified points for every orientation summed up over all the modes.

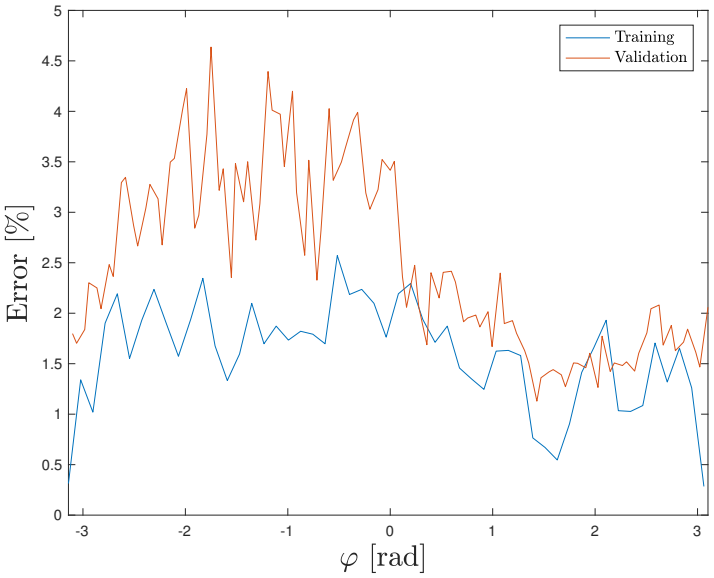

Figure 4.2: Error percentage for different φ

We can see that the error is not homogeneously distributed over the orientation, which makes sense, since the curve has some intrinsic geometric parameters that provoke different changes of the viability kernel for different orientations.

After all, an SVM, that can only classify one curve is not that useful for our application. However, since the SVM creates a continuous classifier over the whole feature space, it is also possible to predict points in between the grid points. This is especially interesting when considering the different velocity modes which replace the velocity dynamics. As the SVM was trained with the features $\bar{v}_x$, $\bar{v}_y$ $\bar{\omega}$ instead of the mode number, we created a way to approximate the viability kernel over all the velocity state space. We can show that with a disturbance on the velocity which could represent the error induced by the mapping onto the closest mode, the SVM classifier seems to adapt the learned safe region in the intuitive correct direction.

Figure 4.3 shows the viability kernel for mode $m = 53$ and $\varphi = 1.5509$ (red and blue points) and the prediction of the SVM with a positive disturbance in the forward velocity e.g. $v_x = \bar{v}_x(m = 53) + d_{v_x}$. The disturbance is chosen such that the new forward velocity lies exactly in-between two modes. One can see that some safe points are no longer in the classified safe regions of the SVM, which makes sense since a greater forward velocity means that the car can turn

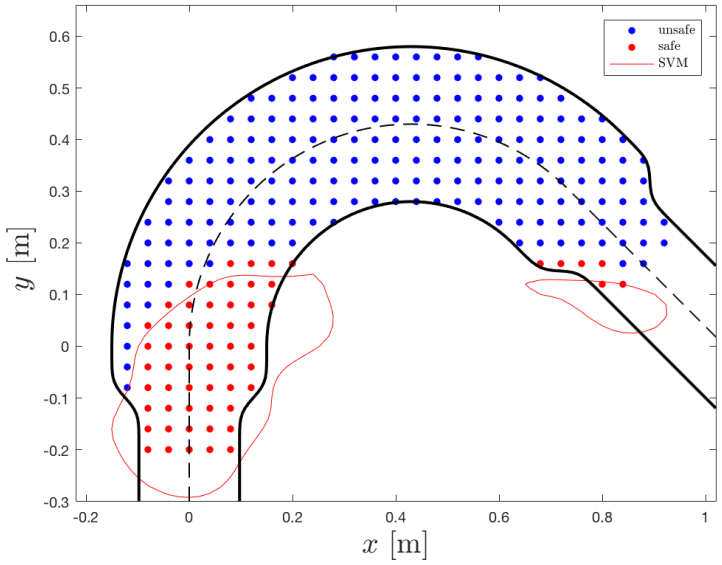

Figure 4.3: SVM in presents of disturbance

less aggressively and possibly violates the track constraints when starting at these critical points. Of course, these are only qualitative results since we do not have the ground truth for that case.

4.2 Single Mode SVM

As described in Section 3.3.5 we trained an SVM for one single velocity mode and a range of different curves. For training, 441 different curves were used with radius ranging from 0.15 m to 0.5 m and the opening angle going from 0 to 2π. Each parameter was gridded uniformly resulting in 21 grid points in each dimension. The parameters for the C-SVM were chosen $\gamma = 10$ and $C = 25$. For cross validation and qualitative results, we chose a velocity mode that is in the average range of the car in real world use: $m = 32$. The training took 253'857 seconds for 2'397'113 training points after reduction SVM classifier seems to adapt the learned safe region in the intuitive correct direction. Figure 4.4 shows a variation of different curves from the training set. We see that the SVM qualitatively performs well without making too many misclassifications.
For a more numerical analysis we check the error percentage, summed up over all orientations for every combination of R and α. Figure 4.5 shows the results for the training data. We cannot see a trend and conclude that the error is mostly homogenous in both parameters.
However, the results we are really interested in are those that are yielded from predicting unknown curves e.g. curves that are not included in the training set. For this purpose we applied the viability algorithm to 25 additional curves that are only used for cross-validation. The curves were generated by choosing a random grid of 5 by 5 in the radius and opening angle while checking the curves

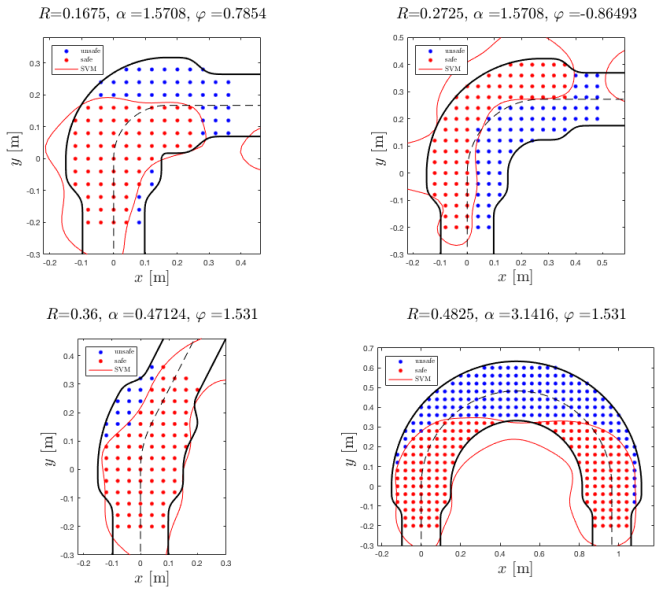

Figure 4.4: Qualitative results of the training set for a single mode SVM

were in the original ranges of R and α. Fig 4.6 shows four of these curves with different orientations.
Qualitatively, one can barely see any change from the results of the trained curves. However, when making the same qualitative analysis as in the case of the training set, we can see a trend of the error going upwards with increasing radii and increasing opening angles. It is not clear if there is such a relation or if the validation set is too small and the results miss leading.

4.2.1 Time Analysis

Since the real use case considers using the single mode SVM's for the path planner, we tested how long it would take to evaluate a single point. We did 1'000 tests in which normalization and prediction was done and time was kept. We got an average time of 35.8 ms with a standard deviation of 2.4 ms on a Macbook Pro with 2.3 GHz Intel Core i7. Two things to keep in mind here is, firstly, the path planner should ideally run on a micro controller and not on a laptop, and secondly, our tests were done partially in Matlab (Normalization), which is an interpreter based programming language, and LIBSVM (prediction). The latter uses a "mex" interface which is potentially slow, since it gives a constant overhead. If fully deployed and compiled, the evaluation could potentially be significantly faster.

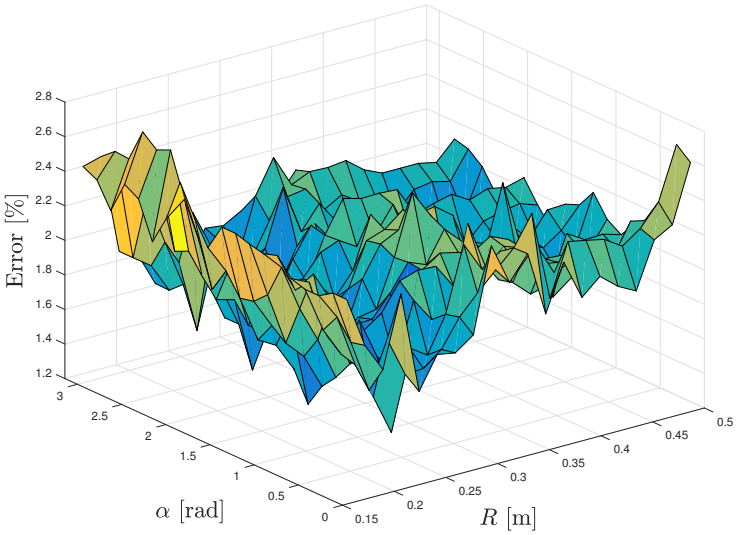

Figure 4.5: Error percentage for the 441 training curves

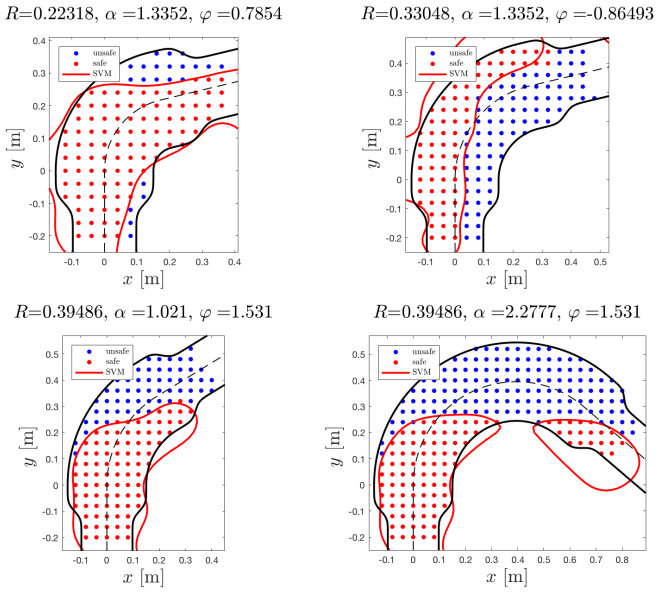

Figure 4.6: Qualitative results of the validation set for a single mode SVM

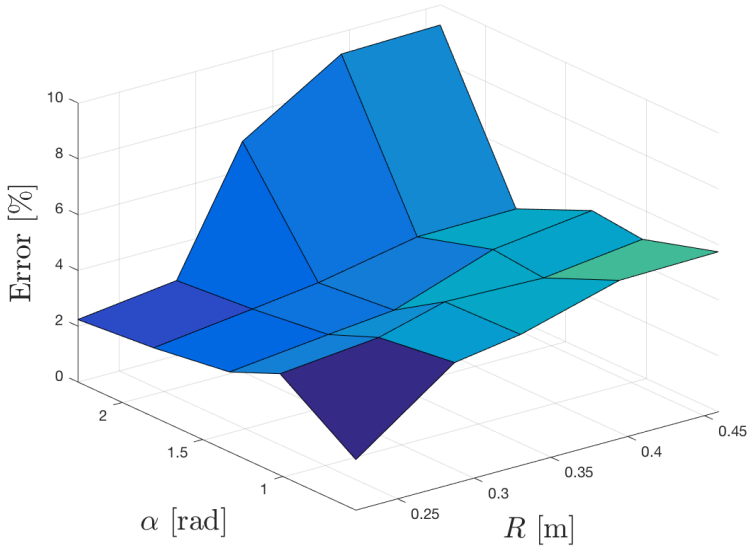

Figure 4.7: Error percentage for the 25 validation curves

Chapter 5

Conclusion

In this thesis we showed that supervised learning can be used to approximate the viability kernel. We showed that SVM is an appropriate tool for such a problem when reducing the number of training points. On the one hand, we showed that an SVM can be trained with the discretized viability kernel and allows us to get a continuous approximation of the viability kernel on the whole state space including the velocities. On the other hand, we showed that for a fixed velocity mode, a new curve can be predicted within a certain error margin with a minimum online computation time. Hence this could be used in a path planner implementation.
Future work would consist in testing the learned SVM on the track and in the controller. Further, we saw that at the moment it is not possible to learn a set of curves with all the features (including velocity modes). In the future, a more detailed investigation in other machine learning tools could be concluded that may be beneficial for the training time.

Bibliography

[1] Orca auronomous racing project. *http://control.ee.ethz.ch/ racing/*.

[2] *Approximation of the viability kernel*, pages 187–209. Springer, 1994.

[3] *Viability Theory*. Springer, 2009.

[4] E. Bakker, L. Nyborg, and H. Pacejka. Tyre modelling for use in vehicle dynamics studies. *SAE*, 1987.

[5] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[6] Laetitia Chapel and Guillaume Deffuant. Svm viability controller active learning: Applicationto bike control. *IEEE Symposium on Approximate Dynamic Programming and Reinfrocment Learning*, April 2007.

[7] Guillaume Deffuant, Laetitia Chapel, and Sophie Martin. Approximating viabilty kernels with support vector machines. *IEEE Transactions on Automatic Control*, 52(5), May 2007.

[8] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, Jul 2014.

[9] Alexander Liniger and John Lygeros. A viability approach for fast recursive feasible finite horizon path planning of autonomous rc cars. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 1–10. ACM, 2015.

[10] Alexander Liniger and John Lygeros. Real-time control for autonomous racing based on viability theory. *arXiv preprint arXiv:1701.08735*, 2017.

[11] P.-O. Vandanjon, A. Coiret, and T.Lorino. Application of viabilty theory for road vehicle active safety during cornering manoeuvres. *Vehicle System Dynamics*, 52(2):244–260, 2014.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Supervised Learning of the Safe Set in Autonomous Driving

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Maessen	Steve

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
12.06.17	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.