A Viability Approach for Fast Recursive Feasible Finite Horizon Path Planning of Autonomous RC Cars

Alexander Liniger
Automatic Control Lab
ETH Zurich
Physikstrasse 3
Zurich, Switzerland
liniger@control.ethz.ch

John Lygeros
Automatic Control Lab
ETH Zurich
Physikstrasse 3
Zurich, Switzerland
lygeros@control.ethz.ch

ABSTRACT

We consider a viability based approach to guarantee recursive feasibility of a finite horizon path planner. The path planner is formulated as a hybrid system for which a difference inclusion reformulation is derived by exploiting the special structure of the problem. Based on this approximation, the viability kernel, which characterizes all safe states and the corresponding safe controls, can be constructed. Using the set of safe controls the computation time of the online path planning can reduced, because only viable trajectories are generated. Finally, a condition characterizing the unsafe set in case of on-line obstacle avoidance is derived.

Categories and Subject Descriptors

J.2 [**Physical Science & Engineering**]: Engineering; I.6.4 [**Simulation & Modeling**]: Model Validation & Analysis

Keywords

Viability Theory, Path Planning, Autonomous Driving, Hybrid Systems

1. INTRODUCTION

Autonomous driving is a challenging task, especially if the car is operated close to the handling limit and the avoidance of obstacles has to be considered, see for example [6, 7]. In [13], optimization-based methods to solve this problem are presented, formulated as a receding finite horizon optimal control problem. Such methods suffers from a common problem in finite receding horizon controller, the lack of recursive feasibility. For problems with state and control constraints it is possible, that decisions of the finite horizon optimization problem steer the closed loop system to states, where constraint violation is inevitable. This problem can be tackled using invariant set theory, as described in [11], where, different kinds of invariant sets are discussed, and necessary and sufficient conditions for feasibility of nonlinear model predictive controllers are derived.

Viability theory, [1], was specifically developed to characterize states where there exist system trajectories that satisfy the constraints forever. Though the use of viability theory in model predictive control has been considered [8] most applications of the theory, normally establish safe/viable regions in the state space. For example in applications such as aerospace [12, 20], underwater vehicles [17, 18], and portfolio management [3]. It had also been used to derive safe feedback controllers, for example in [21].

In [22], a controller guaranteeing safety based on a stochastic reach-avoid problem was successfully implemented on autonomous RC cars, which shows that there is a potential gain by incorporating safe controls in the task of autonomous driving.

In this paper we will explore the use of viability theory for path planning. In particular, we are interested in generating recursively feasible paths on-line, motivated by an automatic driving application. We show that the use of viability theory cannot only avoid recursive feasibility problems, but also speed up the path planning process enabling real time implementation. Related approaches have been reported in [17] where the viability kernel is used to reconstruct safe trajectories and in [10] where learned local viable solutions are used to speed up the path planning, by categorizing dead ends in a given environment/map as non viable.

Our starting point is the path planning algorithm presented in [13], that comprises a path planner and an model predictive controller tracking the output of the path planner. To formalize the process, we formulate the problem in the framework of hybrid automata of [15]. We then reformulate the resulting hybrid system as a discrete time difference inclusion. Based on this, the algorithms presented in [19] are deployed to calculate the viability kernel.

Based on the viability kernel we then propose to reconstruct all safe controls and in this way only generate trajectories which stay in the viability kernel. This reduces the computation time necessary for generating paths. As a consequence, more trajectories and of longer time horizons can be explored in the same amount of time, leading to a better final path. To the best of the authors knowledge this is the first time such an approach is proposed.

Finally, based on the concept of safe controls it is possible to formulate a strong necessary condition to avoid a static obstacle on the track of which the position is not known in advance. This can be done by separating the problem into the sub-problem of staying inside the track constraints, and secondly avoiding static obstacles. Based on the the concept of safe control inputs the separated solutions can be efficiently combined.

In Section 2 the Hierarchical Receding Horizon Controller from [13] is summarized and the path planning problem is extended to multiple segments. In Section 3 the path planning method is formalized as a hybrid system and approximated as a discrete time difference inclusion. In Section 4 the viability kernel is introduced and the algorithm presented in [19] is explained. In Section 5, the viability kernel for the given track is calculated together with the reconstructed safe controls. Based on the safe controls, an improved online path planning algorithm is outlined, which is significantly faster than the standard brute force approach. Finally, a new necessary conditions to avoid a static obstacle with a priori unknown position is presented in Section 6.

2. HIERARCHICAL RECEDING HORIZON CONTROLLER

We consider a car driving along a track, see Figure 1, whose boundaries are known a priori. The hierarchical receding horizon controller presented in [13] for this set-up consists of two levels. At a top level possible trajectories are generated (based on the current state of the car, information about the track and a nonlinear dynamical model of the vehicle) and the one that exhibits the largest progress without leaving the track is selected. In the second level the car model is linearized around the chosen trajectory and the trajectory is tracked using a model predictive controller (MPC).

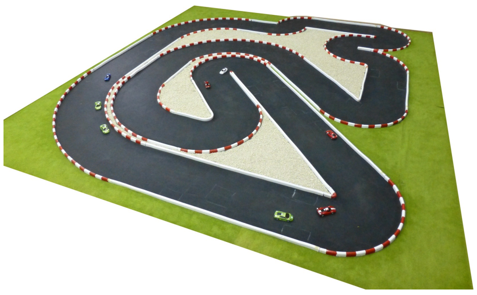

Figure 1: Picture of the track and Kyosho *dnano* cars, used in the experimental set up of [13]. A video of the HRHC can be found at `www.youtube.com/watch?v=ioKTyc9bG4c`

Here we provide a summary of the process and extend it to account for multiple of the so-called zero acceleration points.

2.1 Vehicle Model

The model used for the control design in [13] is a nonlinear bicycle model based on the Pacejka tire model [4]. This model captures the important dynamics, for example the saturation of the nonlinear tire force.

The equations of motion for the car are derived around the center of gravity (CoG), and the states are the position coordinates X and Y, and the orientation φ relative to the inertial frame. These three states characterize the kinematic part of the model. The second part of the model is derived

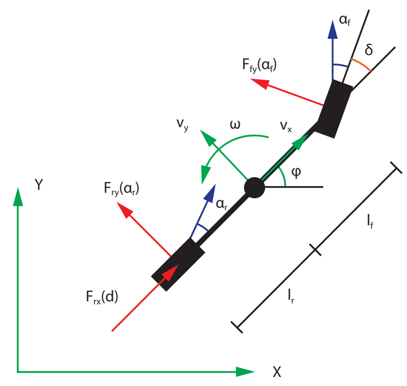

Figure 2: Schematic drawing of the car model

in a body fixed frame centered at the CoG. The states are the longitudinal and lateral velocities v_x and v_y as well as the yaw rate ω. The control inputs are the steering angle δ and the pulse width modulation (PWM) duty cycle d of the drive train motor. The complete equations of motion are

$$\dot{X} = v_x \cos(\varphi) - v_y \sin(\varphi)\,, \tag{1a}$$
$$\dot{Y} = v_x \sin(\varphi) + v_y \cos(\varphi)\,, \tag{1b}$$
$$\dot{\varphi} = \omega\,, \tag{1c}$$
$$\dot{v}_x = \frac{1}{m}(F_{r,x} - F_{f,y}\sin\delta + m v_y \omega)\,, \tag{1d}$$
$$\dot{v}_y = \frac{1}{m}(F_{r,y} + F_{f,y}\cos\delta - m v_x \omega)\,, \tag{1e}$$
$$\dot{\omega} = \frac{1}{I_z}(F_{f,y} l_f \cos\delta - F_{r,y} l_r)\,, \tag{1f}$$

where m is the mass and I_z the moment of inertia of the vehicle. $F_{r,x}(v_x, d)$ is the force produced by 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 rear and the front wheel, calculated using the Pacejka tire model. Lastly, l_r and l_f are the distances from the CoG to the rear and the front wheel respectively.

2.2 Path Planning based on constant velocities

The path planning is based on zero acceleration trajectories, i.e. trajectories where the velocities remain constant (in vehicle dynamics such trajectories correspond to *steady state cornering*, whereas in aeronautics they are referred to as *trimmed flight*). The path planning algorithm of [13] explores a finite number of such zero acceleration trajectories, generated by gridding the stationary velocities for different forward velocities and steering angles. For each, state trajectories are generated by integrating the differential equation of the kinematic model under the assumption that the stationary velocity can be reached instantaneously. Even though this assumption leads to an unrealistic sudden jump in the velocities, it allows one to handle unstable equilibria, for example stationary velocity points representing drifting, which improves maneuverability. Once all the state trajectories have been computed the path planning algorithm processes them, discards the ones that violate the state constraints (e.g. leave the track), and among the remaining ones selects the one that leads to the greatest progress. Where the progress of a considered trajectory is measured by the argument of the projection operator $p : \mathbb{R}^2 \to (0, L]$ which maps any position X, Y, to the arc length θ of the parameterized center line $X^{\text{cen}}(\theta)$, $Y^{\text{cen}}(\theta)$.

$$p(X,Y) := \arg\min_{\theta}(X - X^{\text{cen}}(\theta))^2 + (Y - Y^{\text{cen}}(\theta))^2 \,. \quad (2)$$

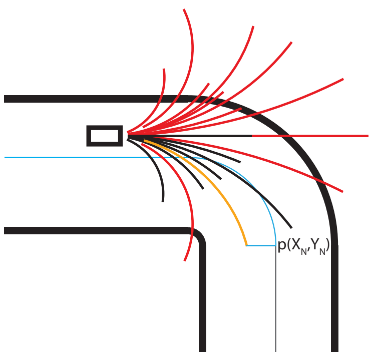

Figure 3: Schematic drawing of the path planning as presented in [13], where the red trajectories get discarded and the orange is the one witht the greatest progerss, which is indicated in blue.

Due to the computational complexity associated with generating, storing and processing the candidate trajectories (many of which will end up being discarded for violating the constraints), only trajectories comprising a single zero acceleration segment were considered in [13]. However, thanks to the computational savings afforded by the viability theory based path planner developed below we are now able to improve the performance considering trajectories comprising multiple zero acceleration segments each of duration T_{pp}.

To ensure that consecutive zero acceleration segments are compatible with each other (and to limit the growth of the search tree which would otherwise be exponential in the number of gridded constant velocity points) we only consider transitions which are achievable by the nonlinear dynamical model (v_x, v_y and ω) in a certain small time step T_t. The problem can be posed as a feasibility problem (3) where the goal is to find controls which allow a transition from a constant velocity point i defined as $\bar{v}(i) = (v_x(i), v_y(i), \omega(i))$ to another point j in at most T_t seconds, subject to the vehicle dynamics and the control constraints,

$$\min_{d,\delta,t_f} \; 0 \quad \text{(3a)}$$

$$\text{s.t.}\; v(0) = (v_x(i), v_y(i), \omega(i)) = \bar{v}(i)\,, \quad \text{(3b)}$$

$$v(t_f) = (v_x(j), v_y(j), \omega(j)) = \bar{v}(j)\,, \quad \text{(3c)}$$

$$\dot{v}(t) = \begin{cases} \dot{v}_x &= \frac{1}{m}(F_{r,x} - F_{f,y}\sin\delta + m v_y \omega)\,, \\ \dot{v}_y &= \frac{1}{m}(F_{r,y} + F_{f,y}\cos\delta - m v_x \omega)\,, \\ \dot{\omega} &= \frac{1}{I_z}(F_{f,y} l_f \cos\delta - F_{r,y} l_r)\,, \end{cases} \quad \text{(3d)}$$

$$t_f \in [0, T_t]\,, \quad \text{(3e)}$$

$$d \in [\underline{d}, \bar{d}] \quad \delta \in [\underline{\delta}, \bar{\delta}]\,. \quad \text{(3f)}$$

This problems allows to classify all the transitions from one stationary point to another, which are then marked as allowed transitions. An illustrative example of such allowed transitions is shown in Figure 4, with 9 constant velocity points for 3 grid points in v_x and δ. The allowed transitions from one constant velocity point $\bar{v}(4)$ are shown by arrows.

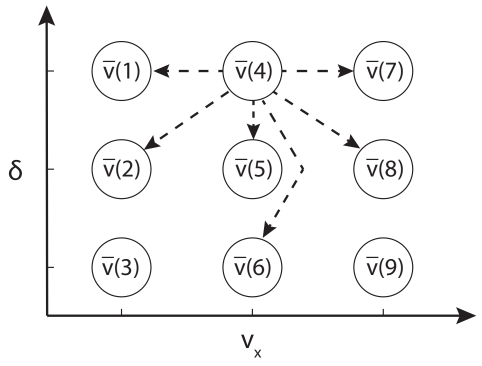

Figure 4: Schematic drowing of allowed transitions from $\bar{v}(4)$ to other constant velocity points.

In practice the problem (3) is solved using the backward reachability algorithm of the multi-parametric toolbox 3.0 [9] using a model linearized around the target velocity and bounds on the controls. The allowed time to reach the new stationary velocity is $T_t = 0.1$s. The constant velocity points are generated by uniformly gridded between $v_x = 0.5$ and 3.5m/s, with 13 grid points, and using between 5 and 11 grid points in δ. This leads to 105 constant velocity points, with about 20 allowed transition at each constant velocity point. Additionally, every discrete velocity level is connected to neighboring levels, similar as shown in Figure 4.

The main challenge for an efficient implementation is the exponential growth of the resulting tree in the number of zero acceleration segments considered. To get an approximation of the complexity, assume there are 20 possible trajectories at each point in the tree. Thus, after 2 levels there are already 400 trajectories. All of those trajectories have to be checked for feasibility with respect to the track and the progress has to be calculated for all trajectories which stay inside the track.

In robotics a similar path planning concept was derived in [5]. However in contrast to the method proposed in [5] and related works such as [7], where different trims are linked by pre-computed maneuvers. We propose to directly link trims and limit the possible transition from one trim to another. Therefore each segment has the same duration T_{pp}, which simplifies the later hybrid system analysis.

3. HYBRID SYSTEM REFORMULATION

The proposed path planning, is similar to a control system with zero order hold and quantized control inputs, which additionally are restricted, based on the last applied control input. Thus, the path planning fits well into a hybrid model framework.

3.1 Path Planner as Hybrid Automaton

The discrete mode q of the hybrid system is the current stationary velocity, and the continuous evolution is given by the kinematic model with the corresponding stationary velocity. Furthermore, a time state is needed to ensure that at every T_{pp} seconds a jump to a different discrete mode/stationary velocity is possible. Thus the continuous evolution is given by,

$$\dot{X} = v_x(q)\cos(\varphi) - v_y(q)\sin(\varphi)\,, \tag{4a}$$

$$\dot{Y} = v_x(q)\sin(\varphi) + v_y(q)\cos(\varphi)\,, \tag{4b}$$

$$\dot{\varphi} = \omega(q)\,, \tag{4c}$$

$$\dot{T} = 1\,, \tag{4d}$$

where $v_x(q), v_y(q), \omega(q)$ are the stationary velocities at the discrete mode q. The discrete transitions takes place every T_{pp} seconds, and a discrete control input determines the discrete mode after the transition. To capture this behavior we consider the following modification of the hybrid automata considered in [15].

Definition 1. *A **hybrid automaton** H is a collection $H = (Q, Z, V, f, D, E, G, R)$ comprising*

- *discrete state variables $q \in Q$,*
- *continuous state variables $z \in Z$,*
- *discrete control inputs $v \in V$,*
- *vector field $f(\cdot,\cdot) : Q \times Z \to Z$,*
- *domain set, $Dom(\cdot) : Q \to 2^Z$,*
- *edges, $E \subset Q \times Q$,*
- *guard, $G(\cdot,\cdot) : E \times V \to 2^Z$,*
- *reset map, $r(\cdot,\cdot) : E \times X \to 2^Z$.*

The main difference to the definitions of [15, 16] is that a discrete input appears in the discrete transition guards. To avoid technical difficulties with the definition of executions of the hybrid automaton given below we introduce the following assumption (adapted from [15]).

Assumption 1. *The cardinality of Q is finite. $Z = \mathbb{R}^n$, for some $n \geq 0$. For all $q \in Q$, the vector field $f(z, q)$ is globally Lipschitz continuous in z. For all $e \in E$, there exist v such that $G(e, v) \neq \emptyset$, and for all, $z \in G(e, v)$, $r(e, z) \neq \emptyset$.*

Definition 2. *[16] A **hybrid time** set $\tau = \{I_i\}_{i=0}^N$ is a finite or infinite sequence of intervals of the real line, such that*

- *$I_i = [\tau_i, \tau_i']$, for all $i < N$,*
- *if $N < \infty$, then either $I_N = \{\tau_N, \tau_N'\}$ or $I_N = \{\tau_N, \tau_N')$,*
- *$\tau_i \leq \tau_i' = \tau_{i+1}$ for all i.*

We are now in a position to define, the execution that the hybrid automaton H accepts. The following definition is adapted from [16].

Definition 3. *[16], Let $\tau = \{I_i\}_{i=0}^N$ be a hybrid time set and consider the sequence of functions $\{q_i(\cdot)\}_{i=0}^N$, $\{z_i(\cdot)\}_{i=0}^N$, $\{v_i(\cdot)\}_{i=0}^N$, with $q_i(\cdot) : I_i \to Q$, $z_i(\cdot) : I_i \to Z$, $v_i(\cdot) : I_i \to V$. The collection of these sequences of functions is called **execution of the hybrid automaton** H starting from initial condition $(q_0(\tau_0), z_0(\tau_0))$, if and only if it satisfies the following conditions:*

- Discrete evolution: *For all $i < N$,*
  1. *$(q_i(\tau_i'), q_{i+1}(\tau_{i+1})) \in E$,*
  2. *$z_i(\tau_i') \in G(q_i(\tau_i'), q_{i+1}(\tau_{i+1}), v_{i+1}(\tau_{i+1}))$,*
  3. *$z_{i+1}(\tau_{i+1}) = r(q_i(\tau_i'), q_{i+1}(\tau_{i+1}), z_i(\tau_i'))$.*
- Continuous evolution: *For all i with $\tau_i < \tau_i'$,*
  1. *$(q_i(t) = q_i(\tau_i))$ and $(v_i(t) = v_i(\tau_i))$ for all $t \in I_i$,*
  2. *$z_i(\cdot) : I_i \to Z$ is the solution of the differential equation*
     $$\dot{z}_i(t) = f(q_i(t), z_i(t)),$$
     over the interval I_i with the initial condition $z_i(\tau_i)$,
  3. *$z_i(t) \in Dom(q_i(t))$ for all $t \in [\tau_i, \tau_i')$.*

The main difference to the corresponding definition of [16] is point 2 in the discrete evolution: The guard depends on $v_{i+1}(\tau_{i+1})$ allowing control of the discrete state after a transition by selecting this variable.

The path planner outlined above can be captured by a hybrid automaton H with,

- $Q = \{1, 2, 3, \cdots, 105\}$,
- $z = (X, Y, \varphi, T) \in \mathbb{R}^4 = Z$,
- $V \subset \mathbb{Z}$,
- $f(z, q)$ given in Eq (4),
- $\text{Dom}(q) = \{(X, Y, \varphi, T) \in \mathbb{R}^4 | T \leq T_{pp}\}$,
- E, all edges where a transition is possible,
- special guard, with two input arguments

$$G(e, v) = \begin{cases} \{z \in \mathbb{R}^4 | T \geq T_{pp}\} & \text{if } G(q_i, q_{i+1}, v) \neq \emptyset \\ \emptyset & \text{else} \end{cases},$$

- $r(e, (X, Y, \varphi, T)) = \{(\hat{X}, \hat{Y}, \hat{\varphi}, \hat{T}) \in \mathbb{R}^4 |$ $\hat{X} = X, \hat{Y} = Y, \hat{\varphi} = \varphi, \hat{T} = 0\}$.

It is easy to see that the constant velocities and the allowed transitions can be always designed such that the path planner automaton satisfies Assumption 1. The Lipschitz condition is easy to verify as the vector field in (4) is differentiable with bounded derivatives.

3.2 Reformulation as a Difference Inclusion

The viability kernel is originally developed for differential inclusions, see [1]. If the system of interest is a hybrid system one would have to resort to sophisticated hybrid viability algorithms [2, 16] to address this problem. In our case, this would dictate gridding the 4 dimensional continuous space, for each of the 105 discrete modes, a task that is computationally very demanding.

Because the discrete transitions take place in regular intervals however, one can reformulate the hybrid dynamics in terms of a differential inclusion, by embedding the discrete state into the real numbers, expressing the discrete dynamics as a transition relation, and writing the sampled data system representation of the continuous dynamics. This then allows the use of the much simpler difference inclusion viability algorithm. For our system this would require gridding a 3 dimensional space, in addition to the coarse grid corresponding to the discrete state.

Let us first define the state space of the difference inclusion, $s = (X, Y, \varphi, q)$ and the states at a specific time step k as $s_k = (X_k, Y_k, \varphi_k, q_k)$. We define a set-valued map $F(s_k)$ as

$$F(s_k) = \begin{bmatrix} F_X(s_k) \\ F_Y(s_k) \\ F_\varphi(s_k) \\ F_q(s_k) \end{bmatrix}, \tag{5}$$

which describes the following discrete time difference inclusion

$$s_{k+1} \in F(s_k)\,. \tag{6}$$

To define the elements of the set-valued map (5), let

$$\begin{bmatrix} X(s_k, t) \\ Y(s_k, t) \\ \varphi(s_k, t) \end{bmatrix}\,. \tag{7}$$

donate the set of solutions of (4a)-(4c) starting at (X_k, Y_k, φ_k), when q is equal to all q_{k+1} such that there exist a $v \in V$ for which $G(q_k, q_{k+1}, v) \neq \emptyset$. Based on this set of solution, let us define,

$$F_X(s_k) = \{X(s_k, T_{pp}\}\,, \tag{8a}$$
$$F_Y(s_k) = \{Y(s_k, T_{pp}\}\,, \tag{8b}$$
$$F_\varphi(s_k) = \{\varphi(s_k, T_{pp}\}\,. \tag{8c}$$

By using the previously defined execution of a hybrid automaton, it is possible to compactly write all possible discrete transitions,

$$F_q(s_k) = \{q_{k+1} \in Q | \exists v \in V :\ \exists G(q_k, q_{k+1}, v) \neq \emptyset\} \tag{9}$$

Thus the path planner is now reformulated as an difference inclusion.

4. VIABILITY THEORY

Having converted our hybrid system into a difference inclusion, here we focus on the approach of [19], where an approximation of the viability kernel for difference inclusion is derived which converges if the discretization goes to zero. We briefly review the results from [19] used in our work.

Viability theory answers the question, for which initial conditions inside a closed set K does there exist a solution to a difference inclusion, which remains in K forever.

Therefore, let Ξ be a finite dimensional vector space and let K be a compact subset of Ξ. Finally the dynamical system of interest is the following discrete time difference inclusion,

$$\xi_{k+1} \in F(\xi_k)\,, \quad \forall k \geq 0\,. \tag{10}$$

A solution of (10) is viable, if it stays in K forever. Which can be characterized by,

$$\xi_{k+1} \in F(\xi_k)\,, \quad \forall k \geq 0\,, \tag{11a}$$
$$\xi_0 = \xi \in K\,, \tag{11b}$$
$$\xi_k \in K\,, \quad \forall k \geq 0\,, \tag{11c}$$

In other words, there exist a portion of (10), starting at $\xi_0 \in K$, which remains in K at each step k. The main interest in viability theory is the subset of initial points $\xi_0 \in K$, for which at least one viable solution exist. Let us define a subset D, for which this is true.

Definition 4. *[19] Let $F : \Xi \to \Xi$ be a set valued map. A subset $D \subset \Xi$ is a* ***discrete viability domain*** *of F if;*

$$\forall \xi \in D, \quad F(\xi) \cap D \neq \emptyset \tag{12}$$

Let K be a subset of Ξ. The ***discrete viability kernel*** *of K under F, is the largest closed subset of closed discrete viability domain contained in K and it is denoted by $Viab_F(K)$.*

4.1 Viability Kernel Algorithm

The discrete viability kernel algorithm is a constructive approach to generate the viability kernel. Let us consider a sequence of subsets K^n, defined as follows,

$$K^0 = K\,, \tag{13a}$$
$$K^{n+1} = \{\xi \in K^n | F(\xi) \cap K^n \neq \emptyset\}\,. \tag{13b}$$

We define,

$$K^\infty = \bigcap_{n=0}^{\infty} K^n\,. \tag{14}$$

Theorem 1. *[19] Let $F : \Xi \to \Xi$ be an upper-semi-continuous set-valued map with closed values and let K be a compact subset of $Dom(F)$. Then,*

$$K^\infty = Viab_F(K) \tag{15}$$

The discrete viability kernel algorithm allows theoretically to calculate the exact viability kernel, however, the algorithm cannot be implemented as presented, since it requires storing and manipulating arbitrary sets. To implement the algorithm one has to resort to discretization, giving rise to a finite difference inclusion.

If the the system of interest is a finite difference inclusion it is only necessary that the system is has finite nonempty values, such that the viability algorithm solves the problem exact. If one is interested in approximating a difference inclusion with a finite one, it is necessary to guarantee that the discretized version has nonempty values. This can be achieved by an extension of the difference inclusion by a ball of radius r. If the radius of the ball is larger than the grid spacing, it is guaranteed that discretizaton of this system has nonempty values. One specific choice for r is the exact grid spacing. If the discretization of any set is denoted by subscript h the grid spacing can be defined by $\|\xi - \xi_h\| \leq \alpha(h)$, which for a suitable discretization goes to zero if h goes to zero.

If $F_h^{\alpha(h)}$ is the finite reduction of the set valued map F first inflated with a ball of the size $\alpha(h)$. The viability kernel of this system is given by the viability kernel algorithm as,

$$K_h^{\alpha(h),\infty} := \bigcap_{n=0}^{\infty} K_h^{\alpha(h),n} = Viab_{F_h^{\alpha(h)}}(K_h)\,. \tag{16}$$

Which if the dicretization goes to zero coincides with the true viability kernel

$$\bigcap_{h>0} Viab_{F_h^{\alpha(h)}}(K_h) = Viab_F(K)\,. \tag{17}$$

Furthermore, $Viab_{F_h^r}(K_h)$ is the reduction of the $Viab_{F^r}(K)$ to the finite set Ξ_h.

5. VIABILITY KERNEL FOR THE TRACK

To apply the viability algorithm to the path planner two challenges have to be overcome. The first one, is related to the assumption in Theorem 1, the theorem assumes that $F : \Xi \to \Xi$ is upper-semicontinuous. Due to the finite difference inclusion in (9) given by the mode jumps, this assumption is clearly not fulfilled. However, the finite dynamics are decoupled from the rest of the difference inclusion, and the discrete dynamics are finite valued by construction of the hybrid system as Q is a finite set. Thus, the viability kernel algorithm will converge to the correct kernel. The second part of the difference inclusion describing the continuous evolution of the hybrid system and is continuous in (X, Y, φ) for a fixed discrete mode.

The second problem comes form the large sampling time T_{pp} of the path planner. The resulting difference inclusion only describes the jumps form an initial condition to all possible next positions. The continuous movement in between is completely neglected. However, it is possible that both the start and the end point of a trajectory are within the set K, but the continuous movement leaves and reenters the set. To solve this problem, the viability algorithm is slightly adjusted such that not only at least one of the possible end points of the continuous evolution has to lie in K^n (13) but additionally also the whole continuous evolution has to stay within K.

5.1 Implementation

The goal is to find a viable solution of the path planning algorithm within the track, see Figure 5. Thus, let us define K as,

$$K := \begin{cases} (X,Y) \in \mathcal{X}_{\text{track}}\,, \\ \varphi \in [-\pi, \pi]\,, \\ q \in Q\,. \end{cases} \tag{18}$$

In this way, X, Y are constrained within the track, φ is not constrained, if all orientations are wrapped to $\pm\pi$ and q is constrained within the set of allowed modes.

The space is gridded as follows,

$$X = [-1.15, 1.8]\text{m with 74 points}\,, \tag{19a}$$
$$Y = [-1.9, 1.7]\text{m with 91 points}\,, \tag{19b}$$
$$\varphi = [-\pi, \pi] \text{ with 84 points}\,, \tag{19c}$$
$$q = \{1, 2, \cdots, 105\}\,. \tag{19d}$$

The grid is chosen, such that the whole track is included, while at the same time the resolution of the grid is as fine as possible with a limited number of grid points. The spatial discretizations has has a resolution of 4 cm which is rather coarse compared to the size of the car which is 12 × 5 cm. However, this already leads to 59,393,880 grid points.

The resulting viability kernel is hard to visualize, as projections down to only X, Y normally fill the whole track. This hides the interesting influence of the angle, and the current discrete mode, which corresponds to the velocity. Therefore, in Figure 5, the viability kernel is only visualized for a fixed angle $\varphi = 30.5°$, and for one discrete mode, which corresponds to the car driving straight with 2 m/s. First, it can be seen that a large part of the track is not viable. However, this is expected, as we look at a specific velocity and angle, which is not suited for many positions. Interesting to see is that this velocity and angle is viable in regions in front of some curves which indicates that this could be a good velocity to drive through the curve.

5.2 Safe Controls

Controlled systems, such as $s_{k+1} = f(s_k, v_k)$, can be reformulated as a difference inclusion of the form,

$$s_{k+1} \in \{f(s_k, v) \mid v \in \mathcal{V}\}\,, \tag{20}$$

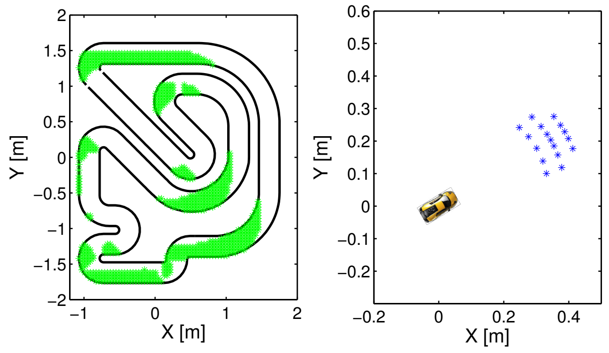

Figure 5: In the left figure, viability kernel for $\varphi = 30.5°$ and the car driving straight with 2 m/s, $q = 44$, and in the right figure, visulaization of the difference inclusion for the given angle and mode.

where $\mathcal{V}$ is the set of allowed controls. For such difference inclusions, once the viability kernel has been computed it is possible to reconstruct safe control inputs that keep the state in the kernel. The set of safe controls for a finite discretization of $s \in S$ denoted by $s_h \in S_h$ is given by,

$$\mathcal{V}_{\text{safe}}(s_h) = \begin{cases} \left\{ \begin{array}{l} v \in \mathcal{V}\,, s_h \in S_h \mid \\ f(s_h, v) + \alpha(h)\mathbb{B} \cap \\ Viab_{F_h^{\alpha(h)}}(K_h) \neq \emptyset \end{array} \right\} & \text{if } s_h \in Viab_{F_h^{\alpha(h)}}(K_h) \\ \emptyset & \text{otherwise} \end{cases} . \tag{21}$$

For this approach to work, $\mathcal{V}$ has to be a finite set, as it is given in model (9). In the general case a new reduction of $\mathcal{V}$ to a finite set has to be introduced.

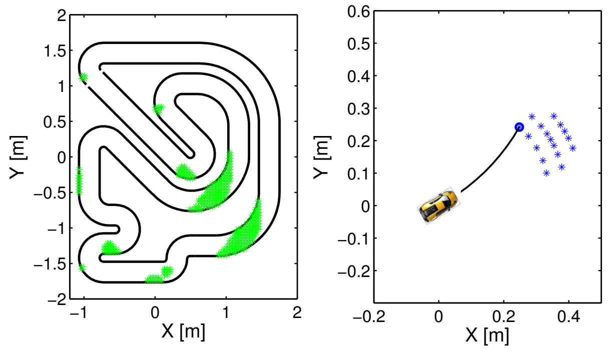

Figure 6: In the left figure, visualization of all possitions, where one control, corresponding to breaking and doing a left turn is safe. For $\varphi = 30.5°$ and the car driving straight with 2 m/s, $q = 44$, and in the right figure visualization of the difference inclusion, and highlighting the control of interest.

Figure 6, shows the same case as Figure 5, however, now only the positions where the control relating to breaking and going left is a safe control. It is clearly visible that this is a subset of the viability kernel visualized in Figure 5.

5.3 Fast Recursive Feasible Path Planning

All the calculation of the viability kernel and the safe controls can be done off-line, where computation times are not critical. The path planning as it is presented in [13] is done online, where the computation time is crucial.

As mentioned in the introduction, the path planner presented in [13] has no recursive feasibility guarantees. The resulting problem can be seen in Figure 7 where the best trajectory is not recursive feasible.

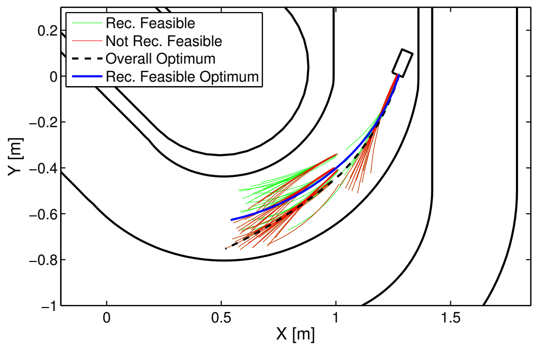

Figure 7: All possible trajectories which stay inside the track, in green all trajectories which stay in the viability kernel, in red all the trajectroies which are not recursive feasible, and in doted black the best trajectory of all trajectories and in blue the best recursive feasible trajectory.

Having the viability kernel there are two different ways to guarantee recursive feasibility. The first and more obvious is to use the viability kernel as a terminal set constraint, an approach typically adopted with invariant sets in MPC. In our setting this would require one to generate a large number of candidate trajectories, then prune the ones that find themselves outside the kernel at the end of the horizon. In turn this means that computation is wasted on generating trajectories that would later on be discarded. An alternative, proposed here, is to use the safe controls $\mathcal{V}_{\text{safe}}(s_h)$ to generate only trajectories that stay in the kernel throughout.

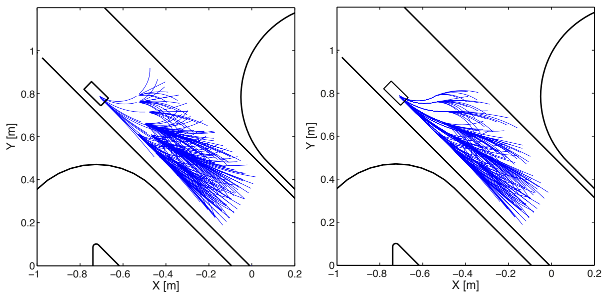

Figure 8: Visualization of a comparision between the naivly generated trajectories on the left, and the trajectories generated using the $\mathcal{V}_{\text{safe}}(s_h)$ for a path planning horizon of two.

Figure 8, shows that the naive approach to generate all trajectories which stay inside the track generates significantly more trajectories, 216 compared to 145 using the proposed trajectory generation using $\mathcal{V}_{\text{safe}}(x_h)$. The naive approach additionally generated a lot of trajectories which get discarded as they leave the track. In total the naive algorithm generates 341 trajectories. All of these have to

Table 1: Computation times of the naive and the viable path planner

	Naive Path Planner	$\mathcal{V}_{\text{safe}}(x)$ Path Planner
mean [s]	0.219	0.0205
max [s]	0.530	0.0749
min [s]	0.034	0.003

be checked with the track constraints, whereas the proposed algorithm does not need any track constraints checks. Note that the additional 71 trajectories of the naive path planner are of no interest as they leave the viability kernel, hence will leave the track in the future.

The proposed path planning strategy reduce the computation time by about one order of magnitude on the average, see Table 5.3. The computation times are generated by starting both path planners at the same position and letting them run for 2000 time steps, which corresponds to multiple laps around the track. The simulations are performed on a MacBook Pro with a 2.4 GHz Intel Core i7 processor, using MATLAB 2013a.

The proposed path generation also improves the quality of the closed loop behavior of the path planner. Lets compare the closed loop behavior of the proposed path planner, with the naive path planner. Two versions of the naive path planner are considered, first one with two constant velocity segments. And secondly one with one constant velocity segment, as implemented in [13], the comparison is done as the computation time is similar to the proposed path planner. The proposed path planner has two constant velocity segments and the same parameters as the first naive path planner.

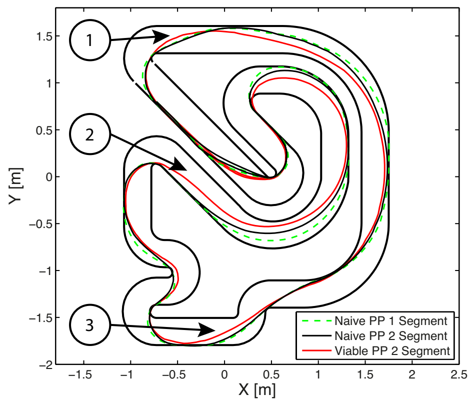

Figure 9: Comparision between closed loop behavior of the naive path planner with one and two segemtns, and the proposed viable path planner

Lets focus on three different positions, marked in Figure 9, where the recursive feasible path planner is significantly better. At location one, the two non-viable path planner start turning too early which later on needs more breaking to be able to perform the sharp turn. In the second location, the naive path planners go close to the constraint, which allows to drive faster, however they end up in position where it is impossible to get around the next turn (only achievable by allowing constraint violations). In the last location the viable path planner starts to move up earlier than the naive path planners, which allows to drive smoother through the chicane and avoid feasibility problems at the second apex.

6. ONLINE OBSTACLE AVOIDANCE

One advantage of on-line generating the possible paths instead of just saving the optimal sequence for each point in S_h is the possibility of incorporating obstacle avoidance. Using viability theory and the idea of reconstructing the safe controls, it is possible to improve the basic approach of finding a path which avoids an obstacle.

The idea is to formulate an obstacle avoidance problem in relative coordinates, where a square obstacle is placed in the origin. Based on this set-up it is possible to calculate the viability kernel, which gives all initial states from where a successful obstacle avoidance is possible. By formulating the problem in relative coordinates, the viability kernel does not depend on the position of the obstacle in the global coordinate system, which allows one to decouple the calculations related to the obstacle avoidance and the track constraint satisfaction.

Figure 10 shows the complement of the viability kernel for obstacle avoidance (unsafe set) in relative coordinates. For every relative orientation this corresponds to a triangular region in front of the obstacle where the collision can no be avoided. The position and the size of the triangle depends on the relative orientation and the speed of the car; Figure 10 shows a slice of the unsafe set for a speed of 2m/s.

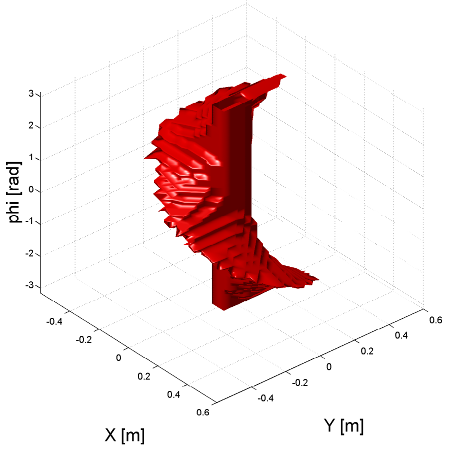

Figure 10: Visualization of the complement of the viability kernel, for the car driving straight with 2 m/s, and an obstacle centered at the origin.

The issue in formulating the obstacle problem in relative coordinates, and decoupling the problem of avoiding an obstacle and staying inside the track is that the individual answers cannot be combined easily. There may exist a certain point s_0 where a solution exists which stays inside the track and one which avoids the obstacle, but not one that achieves both at the same time.

Using the reconstructed safe controls, however, it is possible to formulate necessary conditions for a state to be viable with respect to both the track and the obstacle constraints. Let $\mathcal{V}^{\text{oa}}_{\text{safe}}(s_R)$ be the safe controls, for the obstacle avoidance problems in relative coordinates s_R. Then, a necessary condition that a trajectory is recursively feasible for the combined problem, with the obstacle located at s^{ob}, is that the end point of the trajectory s_N has to fulfill the following condition,

$$\mathcal{V}_{\text{safe}}(s_N) \cap \mathcal{V}^{\text{oa}}_{\text{safe}}(R(\varphi^{\text{oa}})(s_N - s^{\text{ob}})) \neq \emptyset\,, \qquad (22)$$

where $R(\varphi^{\text{oa}})$ is the rotation matrix for a positive ration around φ^{oa}. This is clearly a necessary condition, because if the intersection is empty, there is no control which can guarantee viability with respect to both problems. However, it does not guarantee that by playing an allowed control the intersection at the next time step is also nonempty.

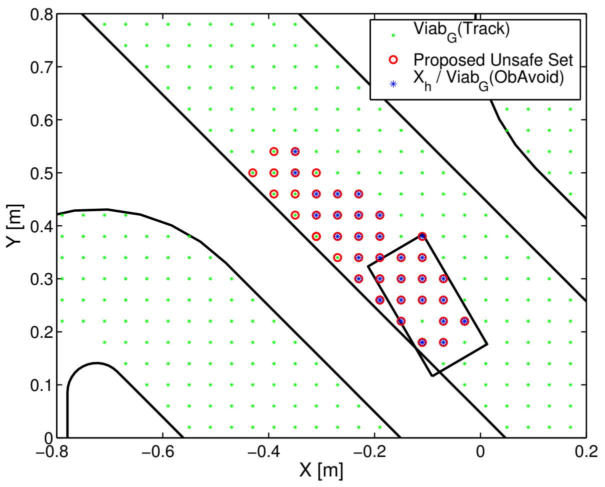

Figure 11: Visualization of the different unsafe sets for an obstacle avoidance problem. Where the car is driving straight with 2m/s parallel to the track, and the obstacle is visualized using a box. In blue stars, the transformed relative obstacle avoidance viability kernel is visualized by projecting it onto S_h and in red cicles the proposed necessary conditon.

In Figure 11, the difference between the presented necessary condition, and the complement of the obstacle avoidance viability kernel are visualized. The car is driving straight with 2 m/s parallel to the track borders and the obstacle is marked with a box. The complement of the viability kernel is visualized with blue stars, and it is clearly visible that there is a dead end below the obstacle. At these positions it is possible to avoid the obstacle on the right side, but, this way is blocked by the track constraints. The proposed necessary condition, (22), can solve this problem and also categorizes this dead end as unsafe. The set of points fulfilling the new condition is clearly a subset of the one lie in the obstacle viability kernel. As the set of safe controls is empty if the state is not in the viability kernel, the intersection with another set of safe controls will also be empty. Thus this condition gives a unsafe set which is at least the size of the complement of the viability kernel.

7. CONCLUSION/FUTURE WORK

In this paper we presented a viability formulation to guarantee the feasibility of a finite horizon path planner. The reconstructed safe controls are successfully used to reduce the online computation time of the path planner by a factor of 10. This is achieved by only generating trajectories which are viable, which allows us to generate fewer trajectories and a priori guarantee that the vehicle stays within the track. Additionally, a new necessary condition is proposed, which establishes a restrictive unsafe region for obstacles with an a priori unknown position, by using the intersection of safe controls of an obstacle avoidance problem formulated in relative coordinates and the global safe controls with respect to the track.

In further work, we try to overcome the limitation that the finite viability kernel is the reduction of the discrete viability kernel. This is not an issue for low dimensional system, where it is possible to solve the viability algorithm for dense grids. However, in higher dimensional application, where the gird is coarse, the information of the grid point cannot be generalized to the $a(h)$ hyper-box around the grid point. In this work, we assumed that this can be done, which sometimes leads to infeasibilities of the path planner.

Additionally, it is also of interest under which conditions the necessary condition (22) is sufficient.

8. REFERENCES

[1] J.-P. Aubin, A. M. Bayen, and P. Saint-Pierre. *Viability Theory, New Directions*. Springer, 2011.

[2] J.-P. Aubin, J. Lygeros, M. Quincampoix, S. Sastry, and N. Seube. Impulse differential inclusions: a viability approach to hybrid systems. *IEEE Transaction on Automatic Control*, 47:2 – 20, 2002.

[3] J.-P. Aubin, D. Pujal, and P. Saint-Pierre. *Dynamic management of portfolios with transaction costs under tychastic uncertainty*. Springer, 2005.

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

[5] E. Frazzoli, M. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, 2005.

[6] Y. Gao, T. Lin, F. Borrelli, E. Tseng, and D. Hrovat. Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads. *Proceedings of DSCC*, 2010.

[7] A. Gray, Y. Gao, T. Lin, J. K. Hedrick, H. E. Tseng, and F. Borrelli. Predictive control for agile semi-autonomous ground vehicles using motion primitives. In *American Control Conference (ACC)*, pages 4239–4244. IEEE, 2012.

[8] L. Grüne and J. Pannek. *Nonlinear Model Predictive Control*. Springer London, 2011.

[9] M. Herceg, M. Kvasnica, C. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013. `http://control.ee.ethz.ch/~mpt`.

[10] M. Kalisiak and M. van de Panne. Faster motion planning using learned local viability models. In *International Conference on Robotics and Automation*, pages 2700 – 2705, 2007.

[11] E. Kerrigan and M. Maciejowski, J. Invariant sets for constrained nonlinear discrete-time systems with application to feasibility in model predictive control. In *Conference on Decision and Control (CDC)*, pages 4951 – 4956, 2000.

[12] I. Kitsios and J. Lygeros. Aerodynamic envelope computation for safe landing of the hl-20 personnel launch vehicle using hybrid control. In *International Symposium on Intelligent Control*, pages 231–236, 2005.

[13] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, pages 1–19, 2014.

[14] J. Lygeros. On reachabiltiy and minimum cost optimal control. *Automatica*, 40:917–927, 2004.

[15] J. Lygeros, K. H. Johansson, S. N. Simic, J. Zhang, and S. S. Sastry. Dynamical properties of hybrid automata. *IEEE Transaction on Automatic Control*, 48:2 – 17, 2003.

[16] K. Margellos and J. Lygeros. Viable set computation for hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 10:45–62, 2013.

[17] R. Moitie and N. Seube. Guidance algorithms for uuvs obstacle avoidance systems. In *OCEANS 2000 MTS/IEEE Conference and Exhibition*, volume 3, pages 1853–1860. IEEE, 2000.

[18] D. Panagou, K. Margellos, S. Summers, J. Lygeros, and K. J. Kyriakopoulos. A viability approach for the stabilization of an underactuated underwater vehicle in the presence of current disturbances. In *Conference on Decision and Control (CDC)*, pages 8612–8617, 2005.

[19] P. Saint-Pierre. Approximation of the viability kernel. *Applied Mathematics and Optimization*, pages 187–209, 1994.

[20] N. Seube, R. Moitie, and G. Leitmann. Aircraft take-off in windshear: a viability approach. *Set-Valued Analysis*, 8(1-2):163–180, 2000.

[21] A. Tinka, S. Diemer, L. Madureira, E. B. Marques, J. B. De Sousa, R. Martins, J. Pinto, J. E. Da Silva, A. Sousa, P. Saint-Pierre, and A. M. Bayen. Viability-based computation of spatially constrained minimum time trajectories for an autonomous underwater vehicle: Implementation and experiments. In *Conference on Decision and Control (CDC)*, pages 3603–3610, 2009.

[22] T. A. Wood, P. Mohajerin Esfahani, and J. Lygeros. Hybrid modelling and reachability on autonomous rc-cars. In *Analysis and Design of Hybrid Systems*, pages 430–435, 2012.