Incremental Sampling-based Algorithm for Minimum-violation Motion Planning

Luis I. Reyes Castro Pratik Chaudhari Jana Tůmová Sertac Karaman Emilio Frazzoli Daniela Rus

Abstract—This paper studies the problem of control strategy synthesis for dynamical systems with differential constraints to fulfill a given task specification while satisfying a set of safety rules. Particular attention is devoted to task specifications that become feasible only if a subset of the safety rules are violated. The proposed algorithm then computes a control law, that minimizes the level of unsafety for a trajectory that satisfies the given task specification. This problem is motivated by an autonomous car navigating an urban environment while following rules of the road such as “always travel in right lane” and “do not change lanes frequently”. Ideas behind sampling based motion-planning algorithms, such as the Probabilistic Road Map (PRM) and Rapidly-exploring Random Tree (RRT), are employed to incrementally construct a finite concretization of the dynamics as a durational Kripke structure. In conjunction with this, a finite automaton that captures the safety rules is used in order to find an optimal trajectory that minimizes the violation of safety rules. It is shown that the proposed algorithm guarantees asymptotic optimality, i.e., almost-sure convergence to optimal solutions. Moreover, the algorithm is also demonstrated in computational experiments.

I. INTRODUCTION

From avoiding traffic jams in busy cities to helping the disabled and elderly on their daily commute, autonomous vehicles promise to revolutionize transportation. However, as they begin to transition from experimental projects like the DARPA Urban Challenge [1] to sharing road infrastructure with human drivers, we need to ensure that they obey the same rules that human drivers are subject to, e.g., rules of the road and safety rules. These rules, such as “always stay in the right lane” and “do not change lanes”, can typically be expressed in formal languages such as Linear Temporal Logic (LTL) and deterministic µ-calculus.

The general problem of finding optimal trajectories satisfying temporal logic tasks has been studied in a number of recent works such as [2]–[5]. In fact, as [6] points out, one of the main challenges of such approaches is the abstraction of continuous systems into equivalent finite transition systems for control synthesis. Moreover, the quality of these controllers depends upon the abstracted finite transition system, and there is no guarantee that a controller will be found (if one exists), i.e., these algorithms are not complete and cannot be applied to, for example, dynamically changing environments.

On a related note, in the robotics literature, algorithms based on Probabilistic Road Maps (PRMs) and Rapidly-exploring Random Trees (RRTs) have been used to synthesize dynamically-feasible trajectories. Algorithms such as PRM* and RRT* [7] are computationally efficient counterparts of these algorithms that guarantee almost sure asymptotic optimality of the returned trajectories. These algorithms have been primarily used for motion planning, and only recently, they have been adapted to handle complex task specifications given in temporal logics [8].

This work focuses on the case when a task specification is infeasible, unless some of the rules can be temporarily broken. Consider, for example, an autonomous car that must reach its final destination while abiding by the rules of the road, such as avoiding collisions with obstacles and staying in the right lane. However, it is more important to reach the destination than to stay in the correct lane. In fact, the latter rule is temporarily violated when, for instance, there is a blockage on the right lane, such as a disabled car. Motivated by these scenarios, we would like to systematically evaluate control strategies that not only pick the rules to be violated but also quantify the level of unsafety of the trajectory and minimize it. In this context, our work is closest in spirit to [9] and [10], and it extends our previous work in [11], where the problem of minimum-violation control synthesis for a pre-defined discrete transition system was considered.

In this paper, using ideas from sampling-based motion planning algorithms, we concretize a continuous-time dynamical system into a finite durational Kripke structure, which captures the time duration required to transition between states. We leverage automata-based model checking approaches to construct a weighted automaton for a given set of prioritized safety rules, which enables us to quantify the level of unsafety of finite input words as the total weight of an accepting run. We next propose an algorithm, MVRRT* (Minimum-Violation RRT*), that incrementally constructs the product of the Kripke structure and the weighted automaton and returns a trajectory of the dynamical system that, (i) minimizes the level of unsafety among all trajectories that satisfy the task, and (ii) minimizes a given cost function among all trajectories that satisfy (i). We prove that as the number of states of the Kripke structure grows to infinity, the solution converges to the optimal trajectory of the dynamical system that satisfies the same criteria.

This paper is organized as follows. We introduce notation and preliminaries in Section II, followed by the problem formulation in Section III. Sections IV-V discuss details of the proposed algorithm. Simulations motivated by an autonomous car are discussed in Section VI. We conclude with directions for future work in Section VII.
II. PRELIMINARIES

A. Durational Kripke Structures for Dynamical Systems

Let $\Pi$ be a set of atomic propositions. The cardinality and the set of all subset of $\Pi$ are denoted by $|\Pi|$ and $2^{|\Pi|}$, respectively. Let $X \subset \mathbb{R}^d$ and $U \subset \mathbb{R}^m$ be compact sets. Consider a dynamical system given by,

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_{init}$$

(1)

where $x_{init}$ is the initial state. Trajectories of the states and controls are denoted by $x : [0, T] \to X$ and $u : [0, T] \to U$ respectively. In the sequel, we tacitly assume that the function $f(\cdot, \cdot)$ is Lipschitz continuous in both its arguments and $u$ is Lebesgue measurable, to guarantee existence and uniqueness of solutions of Equation (1). For a set of atomic propositions $\Pi$, let $\mathcal{L}_c : X \to 2^{\Pi}$ be a function that maps each state to the atomic propositions that it satisfies.

A timed word $\omega$ over $2^\Pi$ is a finite sequence of elements of the set $2^\Pi \times [0, T]$. In particular, a timed word produced by a trajectory $x : [0, T] \to X$ is a finite sequence $\omega(x) = (\ell_0, d_0), (\ell_1, d_1), \ldots, (\ell_{n-1}, d_{n-1}), (\ell_n, d_n)$, where (i) $\ell_0 = \mathcal{L}_c(x(0))$ and $d_0 = 0$, (ii) for all $1 \leq i \leq n - 1$, we have $\ell_i = \mathcal{L}_c(x(t))$, for all $t \in [t_i, t_{i+1})$, where $t_0 = 0$ and $t_i = t_{i-1} + d_i$, and (iii) $\ell_n = \mathcal{L}_c(x(T))$ and $\sum_{n=0}^{n-1} d_i = T$. A word produced by a trajectory $x : [0, T] \to X$ is a finite sequence $\omega(x) = (\ell_0, d_0), (\ell_1, d_1), \ldots, (\ell_{n-1}, d_{n-1}), (\ell_n, d_n)$ is the timed word produced by $x$.

**Definition 1 (Durational Kripke Structure)** A durational Kripke structure is a tuple $\mathcal{K} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L}, \Delta)$, where

- $S$ is a finite set of states;
- $s_{init} \in S$ is an initial state;
- $\mathcal{R} \subseteq S \times S$ is a deterministic transition relation;
- $\Pi$ is a set of atomic propositions;
- $\mathcal{L} : S \to 2^\Pi$ is a state labeling function;
- $\Delta : \mathcal{R} \to 2^\Pi$ is a function assigning a time duration to each transition.

A trace of $\mathcal{K}$ is a finite sequence of states $r = s_0, s_1, \ldots, s_n$ such that $s_0 = s_{init}$ and $(s_i, s_{i+1}) \in \mathcal{R}$, for all $0 \leq i < n$. It produces a finite timed word $\omega(r) = (\mathcal{L}(s_0), \mathcal{L}(s_1), \ldots, (\mathcal{L}(s_{n-1}), \mathcal{L}(s_n)))$ and a word $w(r) = \mathcal{L}(s_0), \mathcal{L}(s_1), \ldots, \mathcal{L}(s_{n-1})$. Given a word $w(r) = \ell_0, \ldots, \ell_n$, let $I = \{i_0, i_1, \ldots, i_k\}$ be a set of indices, such that $i_0 = 0$, $i_j = i_{j-1} + \ldots = i_{j+k-1} \neq i_{j+k}$, for all $0 \leq j < k - 1$, and $i_k = i_{k+1} = \ldots = i_n$. Define

$$\text{destutter}(w(r)) = \ell_{i_0}, \ell_{i_1}, \ldots, \ell_{i_k}.$$  

The following definition is used to concretize a continuous-time dynamical system defined in Eq. (1) into a durational Kripke structure.

**Definition 2 (Trace-Inclusive Kripke Structure)** A durational Kripke structure $\mathcal{K} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L}, \Delta)$ is called trace-inclusive with respect to the dynamical system in Eq. (1) if (i) $S \subseteq X$, (ii) $s_{init} = x_{init}$, (iii) if $(s_1, s_2) \in \mathcal{R}$, there exists a trajectory $x : [0, T] \to X$ such that $x(0) = s_1$, $x(T) = s_2$, $T = \Delta(s_1, s_2)$ and there exists a $T^* \in [0, T]$, such that $\mathcal{L}_c(x(t)) = \mathcal{L}_c(s_1)$, for all $t \in [0, T^*)$, and $\mathcal{L}_c(x(t)) = \mathcal{L}_c(s_2)$, for all $t \in (T^*, T]$ and, (iv) $\mathcal{L}(s) = \mathcal{L}_c(s)$ for all states $s \in S$. Roughly speaking, $\mathcal{L}_c(x(t))$ changes its value at most once between two states.

The following lemma easily follows from the definition above and connects trajectories of the dynamical system with traces of a durational Kripke structure.

**Lemma 3** For any trace $r$ of a trace-inclusive Kripke structure $\mathcal{K}$, there exists a trajectory of the dynamical system, say $x : [0, T] \to X$, such that, destutter($w(r)$) = $w(x)$.

B. Finite Automata

**Definition 4 (Finite Automaton)** A non-deterministic finite automaton (NFA) is a tuple $A = (Q, q_{init}, \Sigma, \delta, F)$, where

- $Q$ is a finite set of states;
- $q_{init} \in Q$ is the initial state;
- $\Sigma$ is an input alphabet;
- $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic transition relation;
- $F \subseteq Q$ is a set of accepting states.

The semantics of finite automata are defined over finite words produced by durational Kripke structures (see Def. 1). In particular, in this work, the alphabet $\Sigma$ is chosen to be $2^\Pi \times 2^\Pi$. A tuple $\tau = (q_1, (\sigma_1, \sigma_2), q_2) \in \delta$ corresponds to a transition labeled with $(\sigma_1, \sigma_2) \in 2^\Pi \times 2^\Pi$ from a state $q_1$ to state $q_2$. A run $\rho$ of a timed automaton over a finite word $w = \ell_0, \ldots, \ell_n$ is a sequence $q_0, \ldots, q_n$ of states, such that $q_0 = q_{init}$, and there exists a transition $(q_i, (\ell_i, (\ell_{i+1}, q_{init})) \in \delta$, for all $0 \leq i < n$. A word $w$ is accepted if and only if there exists a run $\rho = q_0, \ldots, q_n$ over $w$, such that $q_n \in F$.

The language of an automaton $A$, denoted $L(A)$, is the set of all words accepted by $A$.

An automaton $A$ is called non-blocking if, for all $q \in Q$, and $\ell_1, \ell_2 \in \Sigma$, there exists a transition $(q, (\ell_1, \ell_2), q') \in \delta$. For each automaton $A = (Q, q_{init}, \Sigma, \delta, F)$ that is blocking, there exists a language equivalent non-blocking finite automaton $A' = (Q', q_{init}, \Sigma, \delta', F)$, where $Q' = Q \cup \{q_{new}\}$, and $\delta' = \delta \cup \{(q, (\ell_1, \ell_2), q_{new}) \in \delta \in \Sigma, q \in Q'\}$.

C. Finite LTL

Finite automata can capture a large class of properties that are exhibited by traces of a transition system. However, some specification languages with similar expressive power, such as regular expressions or variants of Linear Temporal Logic (LTL) interpreted over finite runs, provide a more user-friendly means to express these properties. Please see [12], [13] for a thorough exposition to regular expressions and LTL. We demonstrate in Section VI, how desirable properties of the system, e.g., the rules of the road and safety rules, can be conveniently captured by a slight modification of Finite LTL (FLTL-$\neg$) [14] without the next operator, a definition for which is given below.

**Definition 5** A FLTL-$\neg$ formula $\phi$ over the set of atomic propositions $\Pi$ is defined inductively as follows:
1) every pair of atomic propositions, \((a, a') \in \Pi \times \Pi\) is a formula,
2) if \(\phi_1\) and \(\phi_2\) are formulas, then \(\phi_1 \lor \phi_2, \neg \phi_1, \phi_1 \cup \phi_2, G \phi_1,\) and \(F \phi_1\) are each formulas,

where \(\neg\) (negation) and \(\lor\) (disjunction) are standard Boolean connectives, and \(U, G, F\) are temporal operators.

Unlike the well-known LTL (see e.g., [13]), FLTL\(_{\infty}\) is interpreted over finite traces, as those generated by the durational Kripke structure from Def. 1. Informally, \((a, a')\) holds true on a trace \(\ell, t_0, \ell_1, \ldots, \ell_n\) if \(a \in \ell_0\), and \(a' \in \ell_1\). The formula \(\phi_1 \cup \phi_2\) states that there is a future moment when formula \(\phi_2\) is true, and formula \(\phi_1\) is true at least until \(\phi_2\) is true. The formula \(G \phi_1\) holds at all positions of a finite trace, and \(F \phi_1\) states that \(\phi_1\) holds at some future time instance. Just as LTL formulas can be algorithmically translated into automata over infinite words, FLTL formulas can be translated into finite automata [15].

D. Level of Unsafty

Let \(\mathcal{K} = (S, s_{\text{init}}, \mathcal{R}, \Pi, \mathcal{L}, \Delta)\) be a durational Kripke structure. Let \(\mathcal{A} = (Q, q_{\text{init}}, 2^\Pi, \delta, F)\) be the automaton for a safety rule with priority \(\varpi(\mathcal{A})\). In the sequel, we assume that the empty trace of \(\mathcal{K}\) and the empty word \(\epsilon\) always satisfy a safety rule given by any \(\mathcal{A}\), i.e., by convention, an empty trace does not violate any safety rules.

**Definition 6 (Level of unsafety for a safety rule)** Let \(w = \ell_0, \ldots, \ell_n\) be a word over \(2^\Pi\), define
\[
\text{vanish}(w, \{i_1, \ldots, i_k\}) = \ell_0, \ldots, \ell_{i_1-1}, \ell_{i_1+1}, \ldots, \ell_n,
\]
i.e., the finite sequence obtained from \(w\) by erasing states indexed with \(i_1, \ldots, i_k\) where \(\{i_1, \ldots, i_k\} \subseteq \{0, \ldots, n\}\). The level of unsafety \(\lambda(w, \mathcal{A})\) of \(w\) with respect to a safety rule expressed as a finite automaton \(\mathcal{A}\) is defined as follows.
\[
\lambda(w, \mathcal{A}) = \min_{I \subseteq \{0, \ldots, n\} \mid \text{vanish}(w, I) \in L(\mathcal{A})} \sum_{i \in I} \varpi(A).
\]

The level of unsafety for a trace \(r = s_0, \ldots, s_{n+1}\) of a durational Kripke structure is then defined to be
\[
\lambda(r, \mathcal{A}) = \min_{I \subseteq \{0, \ldots, n\} \mid \text{vanish}(w(r), I) \in L(\mathcal{A})} \sum_{i \in I} \Delta(s_i, s_{i+1}) \varpi(A).
\]

Consider a sequence of non-empty sets of safety rules \(\Psi = (\Psi_1, \ldots, \Psi_n)\) with each rule \(\psi_j \in \Psi_i\), for all \(1 \leq i \leq n\) given in the form of a finite automaton \(A_{i,j}\). A priority function \(\varpi : \Psi_i \to \mathbb{N}\) assigns a priority to each rule \(A_{i,j}\) in \(\Psi_i\), for all \(1 \leq i \leq n\). The ordered set \(\Psi\) together with the priority function \(\varpi\) is called a set of safety rules with priorities \((\Psi, \varpi)\).

We now extend the definition of the level of unsafety for a word \(w\) and a trace \(r\) to a set of safety rules with priorities \((\Psi, \varpi)\) as follows.

**Definition 7 (Level of unsafety for a set of rules)** The level of unsafety of a word with respect to a set of rules \(\Psi_i\), \(\lambda(w, \Psi_i)\) and the level of unsafety with respect to a set of rules with priorities \((\Psi, \varpi)\) are defined as,
\[
\lambda(w, \Psi_i) = \sum_{A_{i,j} \in \Psi_i} \lambda(w, A_{i,j}),
\]
\[
\lambda(w, \Psi) = \left(\lambda(w, \Psi_1), \ldots, \lambda(w, \Psi_n)\right)
\]

Level of unsafe for a trace \(r\) of \(K\) with respect to a set of rules with priorities is defined similarly. The standard lexicographic ordering for \(\lambda(r, \Psi)\) is used to compare the level of unsafe for two traces \(r_1, r_2\) of \(K\).

III. Problem Formulation

For a compact set \(S \subseteq \mathbb{R}^d\), define \(s_{\text{init}} \in S\) to be the initial state and a compact subset \(S_{\text{goal}} \subseteq S\) as the goal region. Given the dynamical system in Eq. (1), define a task specification \(\Phi\) to be, “traveling from \(s_{\text{init}}\) to \(S_{\text{goal}}\.” A trajectory of the dynamical system, \(x : [0, T] \to X\) is said to satisfy \(\Phi\), if \(x(0) = s_{\text{init}}\) and \(x(T) \in S_{\text{goal}}\). Similarly, a trace \(r = s_0, \ldots, s_n\) of a trace-inclusive Kripke structure satisfies \(\Phi\) if \(s_0 = s_{\text{init}}\) and \(s_n \in S_{\text{goal}}\). We tacitly assume that this task is feasible.

**Problem 8** Given a dynamical system as shown in Eq. (1), a task specification \(\Phi\), a set of safety rules with priorities \((\Psi, \varpi)\) and a continuous function \(c(x)\) that maps a trajectory \(x\) of the dynamical system to a non-negative cost, find a finite trajectory \(x^* : [0, T] \to X\) producing a timed word \(\omega(x^*) = (\ell_0, a_0), \ldots, (\ell_n, a_n)\) such that,

(i) \(x^*\) satisfies the task specification \(\Phi\),
(ii) \(x^*\) minimizes the level of unsafe, \(\lambda(x^*(w), \Psi)\), among all trajectories \(x'\) that satisfy condition (i),
(iii) \(x^*\) minimizes \(c(x')\) among all trajectories \(x''\) that satisfy conditions (i) and (ii).

The solution of this problem as defined above exists if the task \(\Phi\) is feasible. In this work, we restrict ourselves to minimum-time cost functions, i.e., \(c(x) = \int_0^T 1 dt\). The algorithm described here however applies to a much wider class of functions including discounted cost as well as state and control based cost functions. In order to develop an algorithmic approach for Problem 8, we convert it to the following problem defined on a trace-inclusive durational Kripke structure.

**Problem 9** Given a durational Kripke structure \(K = (S, s_{\text{init}}, \mathcal{R}, \Pi, \mathcal{L}, \Delta)\) that is trace-inclusive for the dynamical system in Eq. (1), a task specification \(\Phi\), a set of safety rules with priorities \((\Psi, \varpi)\) and a cost function \(c(x)\), find a finite trace \(x^* = s_0, s_1, \ldots, s_n\) of \(K\) and a trajectory \(x^* : [0, T] \to X\) such that they satisfy the following conditions,

(i) \(x^*\) satisfies \(\Phi\),
(ii) there exists a set \(\{t_0, t_1, \ldots, t_m\}\) such that \(t_0 = 0, t_n = T, s_i = x^*(t_i)\) and \(\Delta(s_i, s_{i+1}) = t_{i+1} - t_i\) \(\forall i < n\),
(iii) \(x^*\) minimizes \(\lambda(\omega(r'), \Psi)\) among all traces \(r'\) of \(K\) that satisfy conditions (i) and (ii) and,
(iv) \(x^*\) minimizes the cost function \(c(x')\) among all trajectories \(x''\) that satisfy (i) and (ii).
IV. ALGORITHM

This section describes an algorithm for finding minimum-constraint violation trajectories for a dynamical system. We first construct a weighted automaton whose weights are chosen such that the weight of an accepting run equals the level of unsafety of the input word. We then propose an algorithm, based on RRT*, to incrementally construct the product of the durational Kripke structure and weighted automata representing safety rules. Roughly, the shortest path in the product uniquely maps to a trace of the Kripke structure that minimizes the level of unsafety. Let us note that the algorithm returns a trajectory that satisfies all rules and minimizes the cost function if it is possible to do so.

A. Weighted product automaton

First, we propose a method to augment each automaton $A_{i,j} \in \Psi$ with new transitions and weights, such that the resulting weighted automaton also accepts all words $w$ that do not satisfy the rule $A_{i,j}$; however, the weights are picked such that the weight of this accepting run over $w$ determines the level of unsafety of $w$ with respect to $A_{i,j}$ (see Def. 10). Second, we combine all the weighted automata into a single weighted automaton $\overline{A}_\Psi$, using its weights to capture the level of unsafety of all words with respect to the given set of safety rules with priorities $(\Psi, \omega)$ (see Def. 12). Third, we build a weighted product of the durational Kripke structure $K$ and the automaton $\overline{A}_\Psi$ (see Def. 14) with the weights corresponding to the level of unsafety of traces of $K$.

We now proceed to describe each of these steps in detail and summarize the purpose of each construction in a lemma (see Def. 10–14 and Lem. 11–15). The material presented in this section is a slight modification of our earlier algorithm for finding a trace of a weighted transition system that minimizes the level of unsafety [11]. For the sake of brevity, proofs of these lemmas are omitted and can be found in [11].

Definition 10 (Automaton $\overline{A}_{\psi, i}$) For a non-blocking finite automaton $A_{i,j} = (Q_{i,j}, q_{init,i,j}, 2^{|Q|}, \delta_{i,j}, F_{i,j})$, a weighted finite automaton

$$\overline{A}_{i,j} = (Q_{i,j}, \eta_{init,i,j}, 2^{|Q|}, \delta_{i,j}, F_{i,j}, W_{i,j})$$

is such that, $\overline{Q}_{i,j} = Q_{i,j}$, $\eta_{init,i,j} = q_{init,i,j}$, $\delta_{i,j} = \delta_{i,j} \cup \{(q, (\sigma, \sigma'), q') \mid q \in Q \land (\sigma, \sigma') \in 2^{|Q|}\}$, $F_{i,j} = F_{i,j}$ and,

$$W(\tau) = \begin{cases} 0 & \text{if } \tau \in \delta_{i,j} \\ \omega(A_{i,j}) & \text{if } \tau \in \delta_{i,j} \setminus \delta_{i,j}. \end{cases}$$

Lemma 11 If a word $w$ over $2^{|Q|}$ is accepted by $A_{i,j}$, then, the weight of the shortest accepting run over $\overline{A}_{i,j}$ over $w$ is equal to the level of unsafety $\lambda(w, \psi_{i,j})$.

A single weighted automaton $\overline{A}_\Psi$ is created by combining all automata $\overline{A}_{i,j}$, where $A_{i,j} \in \Psi, \psi \in \Psi$. This captures the level of unsafety with respect to the whole set of safety rules with priorities $(\Psi, \omega)$ through its weight function.

Definition 12 (Automaton $\overline{A}_\Psi$) The weighted automaton $\overline{A}_\Psi = (Q, \eta_{init}, 2^{|Q|}, C, \delta, F, W)$

is defined as follows:

- $Q = Q_1 \times \cdots \times Q_n$;
- $\eta_{init} = (\eta_{init,1}, \cdots, \eta_{init,n,m})$;
- $(p, (\sigma, \sigma'), p') \in \delta$ if
  - $p = (q_1, \ldots, q_n,m)$, $p' = (q_1', \ldots, q_n,m)$, and
  - $(q_{i,j}, (\sigma, \sigma'), q_{i,j}') \in \delta_{i,j}$ for all $i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$.

Then also $W((p, (\sigma, \sigma'), p')) = (x_1, \ldots, x_n)$, where $x_i = \sum_{j=1}^{m_i} W_{i,j}(q_{i,j}, (\sigma, \sigma'), q_{i,j}')$,

- $F = \{(q_1, \ldots, q_n,m) \mid q_{i,j} \in F_{i,j}, \text{ for all } i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}\}$

Lemma 13 Any word $w$ over $2^{|Q|}$ is accepted by $\overline{A}_\Psi$ and the weight of the shortest accepting run of $\overline{A}_\Psi$ over $w$ is equal to the level of unsafety $\lambda(w, \Psi)$.

Definition 14 (Weighted product automaton $P$) We build the weighted product automaton,

$$P = K \otimes \overline{A}_\Psi = (Q_P, \eta_{init,P}, P, F_P, W_P)$$

of the Kripke structure $K = (S, s_{init}, R, \Pi, C, \Delta)$ and the augmented automaton $\overline{A}_\Psi = (Q, \eta_{init}, 2^{|Q|}, C, \delta, F, W)$ as,

- $Q_P = S \times Q$ is a set of states;
- $\eta_{init,P} = (s_{init}, \eta_{init})$ is the initial state;
- $\delta_P \subseteq Q_P \times Q_P$ is a non-deterministic transition relation, where $((s,q),(s',q')) \in \delta_P$ if $(s,s') \in R$, and there exists a transition $(q, (L(s), L(s'))) \in \delta$.

Then also,

$W_P((s,q),(s',q')) = (x_1 \cdot \Delta(s,s'), \ldots, x_n \cdot \Delta(s,s'))$,

where $(x_1, \ldots, x_n) = W(q, (L(s), L(s')), q')$ and,

- $F_P = S \times F$ is a set of accepting states.

The alphabet of the product automaton is trivial, and therefore omitted. A product automaton is in fact, a finite automaton extended with weights. A run of a product automaton is a sequence $\rho = p_0 \cdots p_n$, such that $p_0 = \eta_{init,P}$, and $(p_i, p_{i+1}) \in \delta_P$, for all $0 \leq i < n$ and it is accepting if $p_n \in F$. The weight of a run $W_P(\rho)$ is the tuple obtained by component-wise sum of the weights associated with the transitions executed along the run. The shortest run over $w$ is then a run $\rho$ minimizing the weight $W_P(\rho)$ in the lexicographical ordering.

Lemma 15 The shortest accepting run (in the lexicographical ordering with respect to $W_P$), $p_0 \cdots p_n$ of $P$ from the state $p_0 = \eta_{init,P}$ to a state $p_n \in F_P$ projects onto a trace $r = \alpha(p_0 \cdots p_n)$ of $K$ that minimizes the level of unsafety.

B. Incremental Weighted Product Automaton

In this section, we incrementally construct the weighted product automaton (see Def. 14) and maintain the trace that minimizes the level of unsafety for a set of safety rules $\Psi$. A few preliminary procedures of the algorithm are as follows.
1. **Sampling:** The sample procedure samples an independent, identically distributed state \( s \) from a uniform distribution supported over the bounded set \( S \).

2. **Nearest neighbors:** The \( \text{near} \) procedure returns the set,

   \[ S_{\text{near}}(s) = \{ s' \mid ||s' - s||_2 \leq \gamma (\log n/n)^{1/d}; \ s' \in S \} \]

   where \( n = |S| \) and \( \gamma \) is the constant in Thm. 16.

3. **Steering:** Given two states \( s, s' \), the \( \text{steer}(s', s) \) procedure computes the pair \((x, T)\) where \( x : [0, T] \rightarrow X \) is a trajectory such that, (i) \( x(0) = s' \), (ii) \( x(T) = s \) and, (iii) \( x \) minimizes the cost function \( c(x) = T \). If a trajectory \( x \) is found, return true, else return false.

4. **Connecting:** For a state \( s' \in S_{\text{near}} \), if \( \text{steer}(s', s) \) returns \( \text{true} \), for all nodes \( z' = (s', q') \in Q_P \), if \( (z', (s, q)) \) is such that it can be in \( \delta_P \) (see Def. 14), the procedure \( \text{Connect}(s', s) \) adds the state \( z = (s, q) \) to the set \( Q_P \), adds \( (z', z) \) to \( \delta_P \) and calculates \( W_P(z', z) \). If \( s \in S_{\text{goal}} \) and \( q \in F \), it adds \( (s, q) \) to \( F_P \).

5. **Updating costs:** The procedure \( \text{Update}(s) \) updates the level of unsafety \( J_n(z) \) and the cost \( J_t(s) \) from the root for a node \( z = (s, q) \) as shown in Alg. 2 using the sets,

   \[ S_{\text{steer}}(s) = \{ s' \mid s' \in S_{\text{near}}(s) \}; \text{steer}(s', s) \text{ returns } \text{true} \}, \]

   \[ Z_{\text{steer}}(s) = \{ (s', q') \mid s' \in S_{\text{steer}}(s); (s', q') \in Q_P \}. \]

6. **Rewiring:** In order to ensure asymptotic optimality, the \( \text{Rewire} \) procedure recalculates the best parent \( \text{Par}(s') \) for all states \( s' \in S_{\text{near}}(s) \) as shown in Alg. 3. The complexity of this procedure can be reduced by noting that \( s' \) only needs to check if the new sample can be its parent by comparing costs \( J_n, J_t \), otherwise its parent remains the same.

Finally, Alg. 1 creates the weighted product automaton as defined in Def. 14 incrementally. It also maintains the best state \( z^* = (s^*, q^*) \in F_P \). The trace \( r^* = s_0, s_1, \ldots, s_n \) of the Kripke structure \( K \) that minimizes the level of unsafety and is a solution to Problem 9 can then be obtained from \( z^* \) by following \( \text{Par}(s^*) \). Since \( K \) is trace-inclusive, the continuous-time trajectory \( x^* \) can be obtained by concatenating smaller trajectories. Let \( (x_i, T_i) \) be the trajectory returned by \( \text{steer}(s_i, s_{i+1}) \) for all states \( s_i \in r^* \). The concatenated trajectory \( x_i : [0, T_i] \rightarrow X \) is such that \( T = \sum_{i=0}^{n-1} T_i \) and \( x_n(t + \sum_{k=0}^{i-1} T_k) = x_i(t) \) for all \( i < n \).

V. ANALYSIS

In this section, we analyze the convergence properties of Algorithm 1. In particular, we prove that the continuous-time trajectory \( x_n \) given by the algorithm after \( n \) iterations converges to the solution of Problem 8 as the number of states in the durational Kripke structure \( K_n \) goes to infinity, with probability one. An analysis of the computational complexity of the algorithm is also carried out here. Due to lack of space, we only sketch the proofs.

A. Asymptotic Optimality

**Theorem 16** The probability that Algorithm 1 returns a durational Kripke structure \( K_n \) and a trajectory of the dynamical system \( x_n \), that converges to the solution of

\[
\mathbb{P} \left( \lim_{n \to \infty} ||x_n - x^*||_{BV} = 0 \right) = 1
\]

**Proof:** (Sketch) The prof primarily follows from the asymptotic optimality of the RRT algorithm (see Theorem 34 in [7]). Let \( x^* : [0, T] \rightarrow X \) be the solution of Problem 8 that satisfies the task \( \Phi \) and minimizes the level of unsafety. For a large enough \( n \), define a finite sequence of overlapping balls \( B_n = \{ B_{n,1}, \ldots, B_{n,m} \} \) around the optimal trajectory \( x^* \). The radius of these balls is set to be some fraction of \( \gamma (\log n/n)^{1/d} \) such that any point in \( s \in B_{n,m} \) can connect to any other point \( s' \in B_{n,m+1} \) using the \( \text{steer}(s, s') \) function. It can then be shown that each ball in \( B_n \) contains at least one state of \( K_n \) with probability one. In such a case, there exists a trace \( r_n = s_0, s_1, \ldots, s_n \) of \( K_n \) such that every state \( s_i \) lies in some ball \( B_{n,m} \). Also, for a large enough \( n \), the level of unsafety of \( r_n, \lambda(r_n, \Phi) \) is equal to the level of unsafety of the word generated by the trajectory \( x^* \), \( \lambda(x^*, \Phi) \), i.e., \( \text{MVRRRT}^* \) returns the trace with the minimum level of unsafety among all traces of the Kripke.
structure $\mathcal{K}$ satisfying the task $\phi$. Finally, it can be shown that the output of the algorithm, trajectory $x_n$, converges to the optimal solution $x^*$ almost surely as $n \to \infty$.

In this proof, $\gamma > 2(2 + \frac{1}{n})^{1/d} \left(\frac{\mu(S)}{\zeta_d}\right)^{1/d}$, where $\mu(S)$ is the Lebesgue measure of the set $S$, $\zeta_d$ is the volume of the unit ball in $S$ which is of dimensionality $d$.

The following lemma is an immediate consequence of Theorem 16 and the continuity of the cost function $c(x)$.

**Lemma 17** The cost of the solution converges to the optimal cost, $c^* = c(x^*)$, as the number of samples approaches infinity, almost surely, i.e.,

$$
\Pr \left( \lim_{n \to \infty} c(x_n) = c^* \right) = 1.
$$

**B. Computational Complexity**

Let us now comment on the computational complexity of MVRRT$^*$.  

**Lemma 18** The expected computational complexity of MVRRT$^*$ is $O(m^2 \log n)$ per iteration where $m$ is the number of states in the durational Kripke structure $\mathcal{K}$ and $n$ is the number of states of the automaton $\mathcal{A}_\Phi$ (see Def. 12).

This follows from an analysis of Algorithm 1. Note that there are an expected $O(\log n)$ samples in a ball of radius $\gamma(\log n/n)^{1/d}$. The procedure Steer is called on an expected $O(\log n)$ samples while because the automaton $\mathcal{A}_\Phi$ is non-deterministic, the procedure Connect adds at most $m^2$ new states in the product automaton per sample. The procedure Update requires at most $O(m^2 \log n)$ time call. The ReWire procedure simply updates the parents of the $O(\log n)$ neighboring samples which take $O(m^2 \log n)$ time. In total, the computational complexity of MVRRT$^*$ is $O(m^2 \log n)$ per iteration.

**VI. COMPUTATIONAL EXPERIMENTS**

In this section, we consider an autonomous car modeled as a Dubin’s vehicle in a road-like environment along with road-safety rules and we evaluate the performance of MVRRT$^*$ in a number of different situations.

**A. Experimental Setup**

Consider a Dubin’s car, i.e., a curvature-constrained vehicle with dynamics given by the differential equations,

\[ 
\begin{align*}
\dot{x}(t) &= v \cos(\theta(t)) \\
\dot{y}(t) &= v \sin(\theta(t)) \\
\dot{\theta}(t) &= u(t)
\end{align*}
\]

The state of the system is the vector $[x, y, \theta]^T$, and the input is $u(t)$, where $|u(t)| \leq 1$ for all $t \geq 0$. The vehicle is assumed to travel at a constant speed $v$. As shown in [16], time-optimal trajectories for this system in an obstacle-free environment can be easily calculated.

Each state of the Dubin’s vehicle is labeled with the set of atomic propositions that hold true in that state. Namely, the atomic propositions capture, (i) whether the vehicle is present in the right lane of the road, the left lane of the road, or the sidewalk, (ii) whether the vehicle is heading in the correct direction or not, and (iii) whether the center line is dotted or solid.

We therefore partition the working domain $S$ into compact non-empty subsets $S_{obs}$, $S_{sw}$, $S_{rl}$, and $S_{ll}$, where $S_{obs}$ is the union of regions occupied with obstacles, $S_{sw}$ represents the sidewalk and $S_{rl}$, $S_{ll}$ are the right and the left lanes, respectively. In the presence of no obstacles, $S_{obs}$ is empty. These partitions are illustrated in Fig. 1. Based on this partitioning and the current state of the Dubin’s vehicle, we define the set of atomic propositions as,

$$
\Pi = \{sw, rl, ll, dir, dotted, solid\}.
$$

For a state $s \in S$ in the trace-inclusive durational Kripke structure $\mathcal{K}$, the proposition $sw$ is true if $s \in S_{sw}$. Similarly, the atomic proposition $rl$ is true if $s \in S_{rl}$ and $ll$ is true if $s \in S_{ll}$. The propositions $rl$ and $ll$ cannot both be true at the same time. The proposition $dir$ models whether the heading of the car is in the correct direction or not, i.e., $dir$ is true if the state $s$ is such that it is heading forwards and $rl$ is true. Note that $dir$ can be calculated using the state $s$ and the geometry of the road. Furthermore, one of the atomic propositions $dotted$ and $solid$ is always true, depending on the lane on which the vehicle’s state is located. Note that obstacles are not considered while constructing $\Pi$ since we do not desire a trajectory that goes over an obstacle in any case. Hence, the $\textbf{Steer}$ procedure in Section IV returns false if any state along the trajectory lies in $S_{obs}$. This change does not affect the correctness and the overall complexity of MVRRT$^*$.

**B. Safety Rules**

Given a task $\Phi$ such as finding a trajectory from $s_{\text{init}}$ to the goal region $S_{\text{goal}}$, we require the vehicle to follow the following rules: (i) do not travel on the sidewalk (sidewalk rule), (ii) do not cross the solid center line (hard lane changing), (iii.a) always travel in the correct direction (direction rule), and (iii.b) do not cross the dotted center line (soft lane changing).

We describe the rules with the following FLTL formulas and corresponding finite automata in Fig. 2. Note that we use 2-tuples of atomic propositions from $\Pi$ as the alphabet for
both formulas and the automata, so that we can specify not only properties of individual states, but also of transitions. The first component of the tuple captures the atomic proposition true in the starting state of the transition; the second component represents the atomic proposition of the ending state.

(i) **Sidewalk**: Do not take a transition that ends at a state labeled \(sw\).

\[ \psi_{1,1} = G \bigwedge_{s \in 2^\mathbb{N}} \neg (s, sw) \]

(ii) **Hard lane change**: Do not take any transition that changes lanes and crosses a solid center line.

\[ \psi_{2,1} = G \left( \neg ((rl, solid) \land (rl, ll)) \lor ((ll, solid) \land (ll, rl)) \right) \]

(iii.a) **Direction**: Do not take any transition that leads to motion in the wrong direction.

\[ \psi_{3,1} = G \bigvee_{s \in 2^\mathbb{N}} (s, dir) \]

(iii.b) **Soft lane change**: Do not take any transition that changes lanes and crosses a dotted center line.

\[ \psi_{3,2} = G \left( \neg ((rl, dotted) \land (rl, ll)) \lor ((ll, dotted) \land (ll, rl)) \right) \]

The finite automata for rules (i)-(iii.b) are all of the same form (see Fig. 2).

Fig. 2: Direction rule. For brevity of presentation, there is only one transition depicted representing in fact all the transitions, where (i) \( l, l' \subseteq 2^\mathbb{N} \), \( l \) is arbitrary and \( dir \in l' \), (ii) \( l, l' \subseteq 2^\mathbb{N} \), such that \( rl \in l \) and \( dotted, ll \in l' \), or \( ll \in l \) and \( dotted, rl \in l \), (iii) \( l, l' \subseteq 2^\mathbb{N} \), such that \( rl \in l \) and \( solid, ll \in l' \), or \( ll \in l \) and \( solid, rl \in l \), and (iv) \( l, l' \subseteq 2^\mathbb{N} \), \( l \) is arbitrary and \( sw \notin l' \).

While it is quite natural to disobey the direction and the soft lane change rules, the solid lane should not be crossed and especially, the sidewalk should be entered only if there is no other way to reach the target region. This gives three different priority classes

\[(\Psi_1, \Psi_2, \Psi_3, \varpi) = (\{\psi_{1,1}\}, \{\psi_{2,1}\}, \{\psi_{3,1}, \psi_{3,2}\}, \varpi),\]

where \(\varpi(\psi_{1,1}) = \varpi(\psi_{2,1}) = \varpi(\psi_{3,1}) = 1\) and \(\varpi(\psi_{3,2}) = 10\). Note that costs for \(\psi_{2,1}\) and \(\psi_{3,2}\) are incurred only once per crossing and do not depend upon the duration of the transition. Within the third class, we put higher priority on the soft lane change rule to avoid frequent lane switching, for instance in case two obstacles are very close to each other and it is not advantageous to come back to the right lane for a short period of time, e.g., see Fig. 4.

C. **Examples**

MVRRT\(^*\) was implemented in C++ on a 2.2GHz processor with 4GB of RAM for the experiments in this section. We demonstrate the performance of MVRRT\(^*\) in a number of different scenarios in the same environment to be able to quantitatively compare the performance. In Fig. 4, the

Dubin’s vehicle starts from the lower right hand corner while the goal region marked in green is located in the lower left hand corner. Light grey denotes the right and left lanes, \(S_{rl}\) and \(S_{ll}\). A sidewalk \(S_{sw}\) is depicted in dark grey. The dotted center line is denoted as a thin yellow line while solid center lines are marked using double lines. Stationary obstacles in this environment are shown in red.

a) **Case 1**: First, we consider a scenario without any safety or road rules. The MVRRT\(^*\) algorithm then simply aims to find the shortest obstacle-free trajectory from the initial state to the goal region. Note, that in this case, MVRRT\(^*\) performs the same steps as the RRT\(^*\) algorithm. The solution computed after 40 seconds has a cost of 88.3 and is illustrated in Fig. 3 together with the adjoining tree.

b) **Case 2**: Next, we introduce the sidewalk rule \(\psi_{1,1}\) and the direction rule \(\psi_{3,1}\). Without any penalty on frequent lane changing, the algorithm gives a trajectory that goes back into the right lane after passing the first obstacle. It has to cross the center line again in order to pass the second obstacle and reach the goal region. Fig 4a depicts the solution obtained after 60 seconds, this has a cost of 122.3 along with a level of unsafety of 46.4 for breaking \(\psi_{3,1}\).

Upon introducing the rule \(\psi_{3,2}\), the vehicle does not go back into the right lane after passing the first obstacle. Figure 4b shows this solution with a level of unsafety of 84.1 for breaking both \(\psi_{3,1}\) and \(\psi_{3,2}\) whereas the level of unsafety in this case for the trajectory in Fig. 4a is 87.4.

In Fig. 4c, we only have the sidewalk and direction rules. Since the road is completely blocked, MVRRT\(^*\) is forced to break the sidewalk rule resulting in a level of unsafety of 139.473 for rules \(\psi_{1,1}\) and \(\psi_{3,1}\), respectively.

c) **Case 3**: Fig 5a shows a run for the sidewalk, direction and the soft lane changing rule after 60 secs. of execution with a level of unsafety of \((0, 0.283)\). As shown in Fig. 5b, if instead executed for 120 secs., the algorithm converges to a solution which has a much higher cost (215.8) but a significantly lower level of unsafety \((0, 1.6)\) because it only breaks the direction rule slightly when it turns into the lane. This thus demonstrates the incrementality and anytime nature of the algorithm.

d) **Case 4**: In our last example, we introduce hard and soft lane changing rules along with sidewalk and direction rules. After 15 secs., MVRRT\(^*\) returns the solution shown
in Fig. 5c, which breaks the hard lane changing rule twice, thereby incurring a level of unsafety of \((0, 2, 48.1)\) for the three rules. On the other hand, after about 300 secs., the solution converges to the trajectory shown in Fig. 5d which breaks the hard lane changing rule only once, this has a level of unsafety of \((0, 1, 25.17)\).

VII. CONCLUSIONS

Motivated by problems in autonomous vehicle navigation, this paper considered the problem of synthesizing minimum-violation control strategies for continuous dynamical systems that are subject to a set of safety rules and satisfy a given task specification. In particular, we focus on the case when the task is infeasible given the set of rules. We leverage ideas from sampling-based motion-planning algorithms and automata-based model checking approaches to propose an incremental algorithm to generate a trajectory of the dynamical system that picks which safety rules to violate while at the same time, it also minimizes the level of unsafety.

Future directions for work include investigation of timed-LTL formulae, modeling rules such as yielding at cross-roads as well as an implementation of the proposed algorithm on an experimental autonomous vehicle.

REFERENCES

[1] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Aalbert Huang, Sertac Karaman, et al. A perception-driven autonomous urban vehicle. Journal of Field Robotics, 25(10):727–774, 2008.

[2] Xu Chu Ding, Stephen L Smith, Calin Bela, and Daniela Rus. MDP optimal control under temporal logic constraints. In Proc. of IEEE Conf. on Decision and Control and European Control Conference (CDC-ECC), pages 532–538, 2011.

[3] Paulo Tabuada and George J Pappas. Linear time logic control of discrete-time linear systems. IEEE Transactions on Automatic Control, 51(12):1862–1877, 2006.

Fig. 4: Fig. 4a shows the solution after 60 secs. for the sidewalk and direction rules. Upon introducing the soft lane changing rule in Fig. 4b, the vehicle does not return to the right lane after passing the first obstacle. Since the road is completely blocked in Fig. 4c, breaking the sidewalk rule is unavoidable.

Fig. 5: Fig. 5a and 5b show the solution of MVRRT after 60 and 120 secs, respectively, with the sidewalk, direction and soft lane changing rules. Note that the algorithm converges to a long trajectory which does not break any rules. Fig. 5c shows a solution after 20 secs, which breaks the hard lane changing rule twice. After 120 secs., the algorithm converges to the solution shown in Fig. 5d, which features only one hard lane change and three soft lane changes.

[4] Stephen L Smith, Jana Tumova, Calin Bela, and Daniela Rus. Optimal path planning for surveillance with temporal-logic constraints. The International Journal of Robotics Research, 30(14):1695–1708, 2011.

[5] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, and Calin Bela. Robust multi-robot optimal path planning with temporal logic constraints. In Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), pages 4693–4698, 2012.

[6] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding horizon control for temporal logic specifications. In Proc. of the 13th ACM Int. Conf. on Hybrid systems: Computation and Control, pages 101–110, 2010.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research, 30(7):846–894, 2011.

[8] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In Proc. of American Control Conference (ACC), 2012.

[9] Vasumathi Raman and Hadas Kress-Gazit. Automated feedback for unachievable high-level robot behaviors. In Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), pages 5156–5162, 2012.

[10] Kris Hauser. The minimum constraint removal problem with three robotics applications. In Proc. of Workshop on the Algorithmic Foundations of Robotics (WAFR), 2012.

[11] Jana Tumova, Gavin C. Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Least-violating control strategy synthesis with safety rules. In Proceedings of the 16th ACM international conference on Hybrid systems: computation and control, ACM, 2013. To appear.

[12] Michael Sipser. Introduction to the Theory of Computation. Course Technology, 3rd edition, 2012.

[13] Christel Baier and Joost-Pieter Katoen. Principles of Model Checking. MIT Press, 2008.

[14] Zohar Manna and Amir Pnueli. Temporal Verification of Reactive Systems: Safety. Springer, 1995.

[15] Elsa L. Gunter and Doron Peled. Temporal debugging for concurrent systems. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pages 431–444, Springer-Verlag, 2002.

[16] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. American Journal of Mathematics, pages 497–516, 1957.