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 reachability goal while satisfying a set of safety rules. Particular attention is devoted to goals that become feasible only if a subset of the safety rules are violated. The proposed algorithm computes a control law, that minimizes the *level of unsafety* while the desired goal is guaranteed to be reached. 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 Probabilistic Road Maps (PRMs) and Rapidly-exploring Random Trees (RRTs), are employed to incrementally construct a finite *concretization* of the dynamics as a durational Kripke structure. In conjunction with this, a weighted finite automaton that captures the safety rules is used in order to find an optimal trajectory that minimizes the violation of safety rules. We prove that the proposed algorithm guarantees asymptotic optimality, i.e., almost-sure convergence to optimal solutions. We present results of simulation experiments and an implementation on an autonomous urban mobility-on-demand system.**

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. 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 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 controller synthesis. Moreover, these controllers depend 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 desired goal 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 rules of the road, such as avoiding collisions with obstacles and staying in the right lane. The former should be obeyed at all times while the latter can be violated in order to reach the goal when the right lane is blocked. Motivated by these scenarios, we would like to systematically evaluate control strategies, 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. 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. 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 goal, 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 goes 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 Sec. II, followed by the problem formulation in Sec. III. Sec. IV and Sec.V discuss details of the proposed algorithm. Simulation experiments and results of an implementation on an autonomous urban mobility-on-demand system are presented in Sec. VI.

II. Preliminaries

A. Durational Kripke Structures for Dynamical Systems

For a set of atomic propositions, Π, let the cardinality and the powerset of Π be denoted by $|\Pi|$ and 2^Π, respectively.

*The authors are with the Massachusetts Institute of Technology, Cambridge, MA, USA.

† The author is with KTH ACCESS Linnaeus Center, Royal Institute of Technology, Sweden and was at Masaryk University, Czech Republic when this work was initiated.

Consider a dynamical system given by,

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(0) = x_{\text{init}} \tag{1}$$

where $X \subset \mathbb{R}^d$ and $U \subset \mathbb{R}^m$ are compact sets and x_{init} is the initial state. Trajectories of states and controls are denoted by $x : [0, T] \to X$ and $u : [0, T] \to U$ respectively, for some $T \in \mathbb{R}_{\geq 0}$.

We assume that $f(\cdot, \cdot)$ is Lipschitz continuous in both its arguments and u is Lebesgue measurable, to guarantee existence and uniqueness of solutions of Eqn. (1). Let $\mathcal{L}_c : X \to 2^\Pi$ be a function that maps each state to atomic propositions that are true at that state.

For a trajectory x, let $D(x) = \{t_i \mid \mathcal{L}_c(x(t_i)) \neq \lim_{s \to t_i^-} \mathcal{L}_c(x(s))\}$ be the set of discontinuities of $\mathcal{L}_c(x(\cdot))$. We assume that $D(x)$ is finite for any x. A trajectory $x : [0, T] \to X$ with $D(x) = \{t_1, \ldots, t_n\}$ produces the finite *timed word*

$$\omega(x) = (\ell_0, d_0), (\ell_1, d_1), \ldots, (\ell_{n-1}, d_{n-1}), (\ell_n, d_n),$$

where (i) $\ell_i = \mathcal{L}_c(x(t_i))$, for all $0 \leq i < n$, with $t_0 = 0$ and $d_i = t_{i+1} - t_i$, and (ii) $\ell_n = \mathcal{L}_c(x(t_n))$ and $d_n = T - t_n$. A *word* produced by this trajectory is defined to be the finite sequence $w(x) = \ell_0, \ell_1, \ldots, \ell_{n-1}, \ell_n$.

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 the initial state, $\mathcal{R} \subseteq S \times S$ is a deterministic transition relation, Π is a set of atomic propositions, $\mathcal{L} \colon S \to 2^\Pi$ is a state labeling function and $\Delta \colon \mathcal{R} \to \mathbb{R}_{\geq 0}$ 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* $\omega(r) = (\ell_0, d_0), \ldots, (\ell_n, d_n)$, where $(\ell_i, d_i) = (\mathcal{L}(s_i), \Delta(s_i, s_{i+1}))$, for all $0 \leq i < n$, and $(\ell_n, d_n) = (\mathcal{L}(s_n), 0)$. The *word* produced by r is $w(r) = \ell_0, \ell_1, \ldots, \ell_n$. Given a word $w(r)$, let $I = \{i_0, i_1, \ldots, i_k\}$ be the unique set of indices such that $i_0 = 0$, $\ell_{i_j} = \ell_{i_j+1} = \ldots = \ell_{i_{j+1}-1} \neq \ell_{i_{j+1}}$ for all $0 \leq j \leq k-1$ and $\ell_k = \ell_{k+1} = \ldots = \ell_n$. Define an operator destutter to remove repeated consecutive elements of a timed word as, $\text{destutter}(w(r)) = \ell_{i_0}, \ell_{i_1}, \ldots, \ell_{i_{k-1}}, \ell_{i_k}$. Let $\langle r \rangle$ denote the duration of a trace, i.e., $\langle r \rangle = \sum_{i=0}^{n} d_i$. The following definition is used to concretize a continuous-time dynamical system into a 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) $S \subset X$, (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 $|D(x)| \leq 1$, i.e., $\mathcal{L}_c(x(\cdot))$ changes its value at most once.*

The following lemma then easily follows from the definition above and relates the trajectories of the dynamical system to 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,* $\text{destutter}(w(r)) = w(x)$.

B. Finite Automata

Definition 4 (Finite Automaton) *A non-deterministic finite automaton (NFA) is a tuple $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, F)$, where Q is a finite set of states; $q_{init} \in Q$ is the initial state; Σ 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 this work, the alphabet Σ 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 q_1 to q_2. A run ρ 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 $(q_i, (\ell_i, \ell_{i+1}), q_{i+1}) \in \delta$, for all $0 \leq i \leq n-1$. A word w is accepted iff there exists a run $\rho = q_0, \ldots, q_n$ over w, such that $q_n \in F$ and rejected otherwise. $L(\mathcal{A})$, called as the language of $\mathcal{A}$, is the set of all words accepted by $\mathcal{A}$.

An automaton $\mathcal{A}$ is called *non-blocking* if, for all $q \in Q$, and $\ell_1, \ell_2 \in \Sigma$, there exists a transition $(q, (\ell_1, \ell_2), q') \in \delta$. Let us note that every blocking automaton can be trivially converted to a non-blocking automaton by adding transitions to a new state $q_{new} \notin F$.

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 (see [12], [13] for details). We demonstrate in Sec. VI, how rules of the road and safety rules can be conveniently captured by a slight modification of Finite LTL [14] without the next operator, called $\text{FLTL}_{-\mathsf{X}}$ and defined below.

Definition 5 ($\text{FLTL}_{-\mathsf{X}}$) *A $\text{FLTL}_{-\mathsf{X}}$ formula ϕ over the set of atomic propositions Π is defined inductively as follows:*

1) *every pair of atomic propositions, $(a, a') \in \Pi \times \Pi$ is a formula,*
2) *if ϕ_1 and ϕ_2 are formulas, then $\phi_1 \vee \phi_2$, $\neg \phi_1$, $\phi_1 \,\mathsf{U}\, \phi_2$, $\mathsf{G}\, \phi_1$, and $\mathsf{F}\, \phi_1$ are each formulas,*

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

Unlike the well-known standard LTL (see e.g., [13]), $\text{FLTL}_{-\mathsf{X}}$ is interpreted over finite traces, as those generated by the durational Kripke structure from Def. 1. Informally, (a, a') holds true on a trace $\ell_0, \ell_1, \ldots, \ell_n$ if $a \in \ell_0$, and $a' \in \ell_1$. The formula $\phi_1 \,\mathsf{U}\, \phi_2$ states that there is a future moment when formula ϕ_2 is true, and formula ϕ_1 is true at least until ϕ_2 is true. The formula $\mathsf{G}\, \phi$ states that formula ϕ holds at all positions of a finite trace, and $\mathsf{F}\, \phi$ states that ϕ

holds at some future time instance. An FLTL$_{-X}$ formula can also be algorithmically translated into a finite automata [15].

D. Level of Unsafety

Let $\mathcal{A}$ be the automaton for a safety rule with priority $\varpi(\mathcal{A})$. The priority function $\varpi : \mathcal{A} \to \mathbb{N}$ assigns priorities to each rule $\mathcal{A}$. We assume here that an empty trace by convention always satisfies the safety rule given by any $\mathcal{A}$.

Definition 6 (Level of Unsafety for a safety rule) *Let $w = \ell_0, \ldots, \ell_n$ be a word over 2^Π, for any index set $I = \{i_1, \ldots, i_k\} \subset \{0, \ldots n\}$, define*

$$\text{vanish}(w, \{i_1, \ldots i_k\}) = \ell_0, \ldots, \ell_{i_j - 1}, \ell_{i_j + 1}, \ldots \ell_n,$$

where $1 \le j \le k$, i.e., the finite sequence obtained from w by erasing states indexed with $i_1, \ldots, i_k$. The level of unsafety $\lambda(w, \mathcal{A})$ of w with respect to a safety rule expressed as a finite automaton $\mathcal{A}$ is,

$$\lambda(w, \mathcal{A}) = \min_{I \mid \text{vanish}(w, I) \in L(\mathcal{A})} \sum_{i \in I} \varpi(\mathcal{A}).$$

The level of unsafety for a timed word $\omega(x) = (\ell_0, d_0), (\ell_1, d_1), \ldots, (\ell_{n-1}, d_{n-1}), (\ell_n, d_n)$ produced by a trajectory x of the dynamical system is,

$$\lambda(x, \mathcal{A}) = \min_{I \mid \text{vanish}(w(x), I) \in L(\mathcal{A})} d_i \cdot \varpi(\mathcal{A}).$$

For a trace $r = s_0, \ldots, s_{n+1}$ of the Kripke structure $\mathcal{K}$, it is

$$\lambda(r, \mathcal{A}) = \min_{I \mid \text{vanish}(w(r), I) \in L(\mathcal{A})} \sum_{i \in I} \Delta(s_i, s_{i+1}) \varpi(\mathcal{A}).$$

Consider a sequence of non-empty sets of safety rules $\mathbf{\Psi} = (\Psi_1, \ldots, \Psi_n)$ with each rule $\psi_j \in \Psi_i$, for all $1 \le i \le n$ given in the form of a finite automaton $\mathcal{A}_{i,j}$. The ordered set $\mathbf{\Psi}$ together with the priority function ϖ is called a set of safety rules with priorities $(\mathbf{\Psi}, \varpi)$. 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 $(\mathbf{\Psi}, \varpi)$ 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 Ψ_i, $\lambda(w, \Psi_i)$ and the level of unsafety with respect to a set of rules with priorities $(\mathbf{\Psi}, \varpi)$ are defined as,*

$$\lambda(w, \Psi_i) = \sum_{\mathcal{A}_{i,j} \in \Psi_i} \lambda(w, \mathcal{A}_{i,j}),$$
$$\lambda(w, \mathbf{\Psi}) = \big(\lambda(w, \Psi_1), \ldots, \lambda(w, \Psi_n)\big)$$

Level of unsafety for a trajectory of the dynamical system and a trace r of $\mathcal{K}$ with respect to a set of rules with priorities is defined similarly. The standard lexicographic ordering is used to compare the level of unsafety of two traces r_1, r_2.

III. Problem Formulation

For a compact set $\mathcal{S} \subset \mathbb{R}^d$, define $s_{init} \in \mathcal{S}$ to be the initial state and a compact subset $\mathcal{S}_{goal} \subset \mathcal{S}$ as the goal region. Given the dynamical system in Eq. (1), define a task specification $\mathbf{\Phi}$ to be, "traveling from s_{init} to $\mathcal{S}_{goal}$". The word produced by a trajectory $x : [0, T] \to X$, $w(x) = \ell_0, \ell_1, \ldots, \ell_n$ is said to satisfy the task $\mathbf{\Phi}$ if $\ell_0 = s_{init}$ and $\ell_n \in \mathcal{S}_{goal}$. Similarly, a trace of the Kripke structure, $r = s_0, \ldots, s_n$ satisfies $\mathbf{\Phi}$ if $s_0 = s_{init}$ and $s_n \in \mathcal{S}_{goal}$. We assume in this work that this task is feasible.

Problem 8 *Given a dynamical system as shown in Eq. (1), a task specification $\mathbf{\Phi}$, a set of safety rules with priorities $(\mathbf{\Psi}, \varpi)$ and a continuous function $c(x)$ that maps a trajectory x of the dynamical system to a non-negative cost, find a trajectory $x^* : [0, T] \to X$ producing a timed word $\omega(x^*) = (\ell_0, d_0) \ldots (\ell_n, d_n)$ and a word $w(x)$ such that,*

(i) $w(x)$ satisfies the task specification $\mathbf{\Phi}$,
(ii) x^* minimizes the level of unsafety, $\lambda(x', \mathbf{\Psi})$, 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 $\mathbf{\Phi}$ 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 with minor changes. In order to develop an algorithmic approach for Prob. 8, we convert it to the following problem defined on a trace-inclusive durational Kripke structure. Thm. 16 connects the solutions of Prob. 9 to those of Prob. 8.

Problem 9 *Given a durational Kripke structure $\mathcal{K} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L}, \Delta)$ that is trace-inclusive for the dynamical system in Eq. (1), a task specification $\mathbf{\Phi}$, a set of safety rules with priorities $(\mathbf{\Psi}, \varpi)$ and a cost function $c(x)$, find a finite trace $r^* = s_0, s_1, \ldots, s_n$ of $\mathcal{K}$ such that,*

(i) r^* satisfies $\mathbf{\Phi}$,
(ii) r^* minimizes $\lambda(r', \mathbf{\Psi})$ among all traces r' of $\mathcal{K}$ that satisfy condition (i),
(iii) r^* minimizes $\langle r \rangle$ among all traces r'' satisfying (i), (ii).

IV. Algorithm

This section describes an algorithm for finding minimum-constraint violation trajectories for a dynamical system. We then propose an algorithm, based on RRT*, to incrementally construct a product of the Kripke structure and 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 augment each automaton $\mathcal{A}_{i,j} \in \mathbf{\Psi}$ with new transitions and weights, such that the resulting weighted automaton also accepts all words w that *do not* satisfy the rule $\mathcal{A}_{i,j}$; the weights are picked such that the weight of an accepting run over w determines the level of unsafety of w with respect to $\mathcal{A}_{i,j}$ (see Def. 10). Second, we combine all the weighted automata into a single weighted automaton $\overline{\mathcal{A}}_{\mathbf{\Psi}}$;

the weights of this automaton capture the level of unsafety with respect to a set of safety rules with priorities $(\mathbf{\Psi}, \varpi)$ (see Def. 12). Third, we build the product of the durational Kripke structure $\mathcal{K}$ and the automaton $\overline{\mathcal{A}}_{\mathbf{\Psi}}$ (see Def. 14); weights of this product correspond to the level of unsafety of traces of $\mathcal{K}$.

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 (Weighted Automaton) *For a non-blocking finite automaton $\mathcal{A} = (Q, q_{init}, 2^{\Pi}, \delta, F)$, the weighted finite automaton is defined as $\overline{\mathcal{A}} = (Q, q_{init}, 2^{\Pi}, \overline{\delta}, F, \overline{\mathcal{W}})$, where, $\overline{\delta} = \delta \cup \{(q, (\sigma, \sigma'), q') \mid q, q' \in Q,\ (\sigma, \sigma') \in 2^{\Pi^2}\}$,*

$$\overline{\mathcal{W}}(\tau) = \begin{cases} 0 & \text{if } \tau \in \delta \\ \varpi(\mathcal{A}) & \text{if } \tau \in \overline{\delta} \setminus \delta. \end{cases}$$

Lemma 11 *For a rule $\psi_{i,j}$ given as an automaton $\mathcal{A}_{i,j}$, any word over 2^{Π} is accepted by $\overline{\mathcal{A}}_{i,j}$ and the weight of the shortest accepting run is equal to $\lambda(w, \psi_{i,j})$.*

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

Definition 12 (Automaton $\overline{\mathcal{A}}_{\mathbf{\Psi}}$) *The weighted automaton $\overline{\mathcal{A}}_{\mathbf{\Psi}} = (\overline{Q}, \overline{q}_{init}, 2^{\Pi}, C, \overline{\delta}, \overline{F}, \overline{\mathcal{W}})$ is defined as follows:*

- *$\overline{Q} = Q_{1,1} \ldots \times \ldots Q_{1,m_1} \ldots \times \ldots Q_{n,1} \ldots \times \ldots Q_{n,m_n}$;*
- *$\overline{q}_{init} = (\overline{q}_{init,1,1}, \ldots, \overline{q}_{init,n,m_n})$;*
- *$(p, (\sigma, \sigma'), p') \in \overline{\delta}$ if*
  - *$p = (q_{1,1}, \ldots, q_{n,m_n})$, $p' = (q'_{1,1}, \ldots, q'_{n,m_n})$, and*
  - *$(q_{i,j}, (\sigma, \sigma'), q'_{i,j}) \in \overline{\delta}_{i,j}$, for all $i \in \{1, \ldots, n\}, j \in \{1, \ldots m_i\}$.*

  Also, $\overline{\mathcal{W}}((p, (\sigma, \sigma'), p')) = (x_1, \ldots, x_n)$, where $x_i = \sum_{j=1}^{m_i} \overline{\mathcal{W}}_{i,j}(q_{i,j}, (\sigma, \sigma'), q'_{i,j})$;
- *$\overline{F} = \{(q_{1,1}, \ldots, q_{n,m_n}) \mid q_{i,j} \in \overline{F}_{i,j},$ for all $i \in \{1, \ldots, n\}, j \in \{1, \ldots m_i\}\}$*

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

Definition 14 (Weighted Product Automaton $\mathcal{P}$) *We build the weighted product automaton,*

$$\mathcal{P} = \mathcal{K} \otimes \overline{\mathcal{A}}_{\mathbf{\Psi}} = (Q_{\mathcal{P}}, q_{init,\mathcal{P}}, \delta_{\mathcal{P}}, F_{\mathcal{P}}, \mathcal{W}_{\mathcal{P}})$$

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

- *$Q_{\mathcal{P}} = S \times \overline{Q}$ is a set of states;*
- *$q_{init,\mathcal{P}} = (s_{init}, \overline{q}_{init})$ is the initial state;*
- *$\delta_{\mathcal{P}} \subseteq Q_{\mathcal{P}} \times Q_{\mathcal{P}}$ is a non-deterministic transition relation, where $((s, q), (s', q')) \in \delta_{\mathcal{P}}$ if $(s, s') \in \mathcal{R}$, and there exists a transition $(q, (\mathcal{L}(s), \mathcal{L}(s')), q') \in \overline{\delta}$. Then also,*

  $$\mathcal{W}_{\mathcal{P}}\big((s, q), (s', q')\big) = (x_1 \cdot \Delta(s, s'), \ldots, x_n \cdot \Delta(s, s')),$$

  where $(x_1, \ldots, x_n) = \overline{\mathcal{W}}(q, \mathcal{L}(s), \mathcal{L}(s'), q')$ and,
- *$F_{\mathcal{P}} = (S \cap \mathcal{S}_{goal}) \times \overline{F}$ is a set of accepting states.*

A product automaton is in fact, a finite automaton extended with weights. A run of a product automaton is a sequence $\rho = p_0, \ldots, p_n$, such that $p_0 = q_{init,\mathcal{P}}$, and $(p_i, p_i{+}1) \in \delta_{\mathcal{P}}$, for all $0 \leq i < n$ and it is accepting if $p_n \in F_{\mathcal{P}}$. The weight of a run $\mathcal{W}_{\mathcal{P}}(\rho)$ 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 ρ minimizing the weight $\mathcal{W}_{\mathcal{P}}(\rho)$ in the lexicographical ordering.

Lemma 15 *The shortest accepting run (in the lexicographical ordering with respect to $\mathcal{W}_{\mathcal{P}}$), $p_0 \ldots p_n$ of $\mathcal{P}$ from the state $p_0 = q_{init,\mathcal{P}}$ to a state $p_n \in F_{\mathcal{P}}$ projects onto a trace $r = s_0, \ldots s_n$ of $\mathcal{K}$ 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 $\mathbf{\Psi}$. 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 $\mathcal{S}$.

2) Nearest neighbors: The `Near` procedure returns the set,

$$S_{near}(s) = \{s' | \,||s' - s||_2 \leq \gamma \left(\log n/n\right)^{1/d};\ s' \in S\}$$

where $n = |S|$ and γ is a constant given in Thm. 16.

3) Steering: Given two states s, s', the `Steer`(s', s) procedure computes the pair (x, T) where $x : [0, T] \to X$ 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 $s' \in S_{near}$, if `Steer`(s', s) returns true, for all nodes $z' = (s', q') \in Q_{\mathcal{P}}$, for all $(z', (s, q)) \in \delta_{\mathcal{P}}$, the procedure `Connect`$(s', s)$ adds the state $z = (s, q)$ to the set $Q_{\mathcal{P}}$, adds (z', z) to $\delta_{\mathcal{P}}$ and calculates $\mathcal{W}_{\mathcal{P}}(z', z)$. If $s \in \mathcal{S}_{goal}$ and $q \in F$, it adds (s, q) to $F_{\mathcal{P}}$.

5) Updating costs: The procedure `Update`(s) updates the level of unsafety $J_a(z)$ and the cost $J_t(s)$ from the root for a node $z = (s, q)$ as shown in Alg. 2 using the sets,

$$S_{steer}(s) = \{s' \mid s' \in S_{near}(s);\ \texttt{Steer}(s', s) \text{ returns } \textbf{true}\},$$

$$Z_{steer}(s) = \{(s', q') \mid s' \in S_{steer}(s);\ (s', q') \in Q_{\mathcal{P}}\}.$$

6) Rewiring: In order to ensure asymptotic optimality, the `Rewire` procedure recalculates the best parent `Par`(s') for all states $s' \in S_{near}(s)$ 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_a, 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 $z^* = (s^*, q^*) \in F_{\mathcal{P}}$. The trace $r^* = s_0, s_1, \ldots, s_n$ of the Kripke structure $\mathcal{K}$ that minimizes the level of unsafety and is a solution to Prob. 9 can then be obtained from z^* by following $\mathtt{Par}(s^*)$. Since $\mathcal{K}$ 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 $\mathtt{Steer}(s_i, s_{i+1})$ for all states $s_i \in r^*$. The concatenated trajectory $x^* : [0, T] \to X$ 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 $i < n$.

```
Algorithm 1: Create Product
1  Input : $n$, $\mathcal{S}$, $\overline{\mathcal{A}}_{\Psi}$;
2  $\mathcal{P} \leftarrow \varnothing$; $Q_{\mathcal{P}} \leftarrow q_{\mathcal{P}}^{init}$; $J_a(s_{init}) \leftarrow 0$; $J_t(s_{init}) \leftarrow 0$;
3  $i \leftarrow 0$;
4  for $i \leq n$ do
5      $s \leftarrow$ Sample;
6      for $s' \in$ Near($s$) do
7          if Steer($s', s$) then
8              Connect($s', s$);
9      Par, $J_a, J_t \leftarrow$ Update($s$);
10     $\mathcal{P}, J_a, J_t \leftarrow$ Rewire($s$);
11 $\mathcal{P}_n \leftarrow (Q_{\mathcal{P}}, q_{\mathcal{P}}^{init}, \delta_{\mathcal{P}}, F_{\mathcal{P}}, \mathcal{W}_{\mathcal{P}})$;
12 return $\mathcal{P}_n$
```

```
Algorithm 2: Update($s, \mathcal{P}$)
1  for $z = (s, q) \in Q_{\mathcal{P}}$ do
2      $J_a(z) \leftarrow \min_{z' \in Z_{steer}} \mathcal{W}_{\mathcal{P}}(z', z) + J_a(z')$;
3      $Z^* \leftarrow \arg\min_{z' \in Z_{steer}} \mathcal{W}_{\mathcal{P}}(z', z) + J_a(z')$;
4      $J_t(s) \leftarrow \min_{z' \in Z^*} c(s', s) + J_t(s')$;
5      Par($z$) $\leftarrow \arg\min_{z' \in Z^*} c(s', s) + J_t(s')$;
6  return Par, $J_a, J_t$
```

```
Algorithm 3: Rewire($s, \mathcal{P}$)
1  for $s' \in S_{steer}(s)$ do
2      if Steer($s, s'$) then
3          Connect($s, s'$);
4      $J_a, J_t \leftarrow$ Update($s'$);
5  return $\mathcal{P}$
```

V. Analysis

In this section, we analyze the convergence properties of Alg. 1. In particular, we prove that the continuous-time trajectory x_n given by the algorithm after n iterations converges to the solution of Prob. 8 as the number of states in the durational Kripke structure $\mathcal{K}_n$ goes to infinity, with probability one. A brief analysis of the computational complexity of the algorithm is also carried out here. Due to lack of space, we only sketch the proofs.

Theorem 16 *The probability that Alg. 1 returns a durational Kripke structure $\mathcal{K}_n$ and a trajectory of the dynamical system x_n, that converges to the solution of Prob. 8 in the bounded variation norm sense, approaches one as the number of states in $\mathcal{K}_n$ tends to infinity, i.e.,*

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

Proof: (Sketch) The proof primarily follows from the asymptotic optimality of the RRT* algorithm (see Theorem 34 in [7]). Let $x^* : [0, T] \to X$ be the solution of Prob. 8 that satisfies the task $\mathbf{\Phi}$ 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 $\mathtt{Steer}(s, s')$ function. It can then be shown that each ball in B_n contains at least one state of $\mathcal{K}_n$ with probability one. In such a case, there also exists a trace $r_n = s_0, s_1, \ldots, s_n$ of $\mathcal{K}_n$ 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, $\lambda(r_n, \mathbf{\Psi})$ is equal to the level of unsafety of the word generated by the trajectory x^*, $\lambda(\omega(x^*), \mathbf{\Psi})$, i.e., MVRRT* returns the trace with the minimum level of unsafety among all traces of the Kripke structure $\mathcal{K}$ satisfying the task ϕ. Finally, it can be shown that the trajectory x_n constructing by contanetating smaller trajectories joining consecutive states of r, i.e., $s_0, s_1, \ldots$ converges to x^* almost surely as $n \to \infty$.

In this proof, $\gamma > 2\,(2 + 1/d)^{1/d}\,(\mu(\mathcal{S})/\zeta_d)^{1/d}$, where $\mu(\mathcal{S})$ is the Lebesgue measure of the set $\mathcal{S}$ and ζ_d is the volume of the unit ball of dimensionality d. ■

The following lemma is an immediate consequence of Thm. 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, $\mathbb{P}\left(\{\lim_{n\to\infty} c(x_n) = c^*\}\right) = 1$.*

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

VI. Experiments

In this section, we consider an autonomous vehicle modeled as a Dubins car in an urban environment with road-safety rules and evaluate the performance of MVRRT* in a number of different situations.

A. Experimental Setup

Consider a Dubins car, i.e., a curvature-constrained vehicle with dynamics, $\dot{x} = v\cos(\theta), \dot{y} = v\sin(\theta)$ and $\dot{\theta} = u$. 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

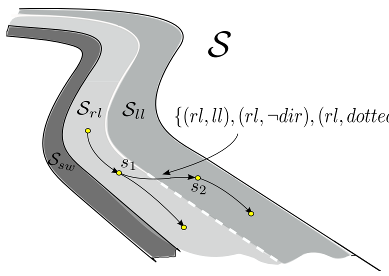


Fig. 1: Partitions of the working domain $\mathcal{S}$. The transition from s_1 to s_2 is labeled with, for example, $\{(rl, ll), (rl, \neg dir), (rl, dotted)\}$.

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.

We partition the working domain $\mathcal{S}$ into compact non-empty subsets $\mathcal{S}_{obs}$ which is the union of obstacled regions, $\mathcal{S}_{sw}$ which represents the sidewalk and $\mathcal{S}_{rl}$, $\mathcal{S}_{ll}$ which are the right and left lanes, respectively, as illustrated in Fig. 1. $\mathcal{S}_{obs}$ is empty if there are no obstacles. Based on this partitioning, we define the set of atomic propositions as, $\Pi = \{sw, rl, ll, dir, dotted, solid\}$. A proposition $p \in \{sw, rl, ll\}$ is true at a state $s \in S$, if $s \in \mathcal{S}_p$ with rl, ll being mutually exclusive. dir is true iff the heading of the car is in the correct direction, i.e., if s is such that the car heading forwards and rl is true. Atomic propositions, $dotted$ and $solid$, depict the nature of lane markers. Note that obstacles are not considered while constructing Π since we do not desire a trajectory that goes over an obstacle. The `Steer` procedure in Sec. IV, instead, returns false if any state along the trajectory lies in $\mathcal{S}_{obs}$. This change does not affect the correctness and the overall complexity of MVRRT*.

B. Safety Rules

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

We describe the rules with the following FLTL$_{-\mathsf{X}}$ formulas and corresponding finite automata in Fig. 2. Note that we use 2-tuples of atomic propositions from Π as the alphabet for both formulas and the automata, to specify not only properties of individual states, but also of transitions. The two components capture the atomic propositions of the starting and the ending state respectively.

(i) Sidewalk: Do not take a transition that ends in $\mathcal{S}_{sw}$.

$$\psi_{1,1} = \mathsf{G} \bigwedge_{* \in 2^\Pi} \neg(*, sw)$$

(ii) Hard lane change: Do not cross a solid center line.

$$\psi_{2,1} = \mathsf{G}\Big(\neg\big((rl, solid) \wedge (rl, ll)\big) \vee \big((ll, solid) \wedge (ll, rl)\big)\Big)$$

(iii.a) Direction: Do not travel in the wrong direction.

$$\psi_{3,1} = \mathsf{G} \bigvee_{* \in 2^\Pi} (*,\ dir)$$

(iii.b) Soft lane change: Do not cross a dotted center line.

$$\psi_{3,2} = \mathsf{G}\Big(\neg\big((rl, dotted) \wedge (rl, ll)\big) \vee \big((ll, dotted) \wedge (ll, rl)\big)\Big)$$

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

$$\begin{tikzcd} {} \arrow[r] & \boxed{q_1} \arrow[loop right, "(\ell,\ell')"] \end{tikzcd}$$

Fig. 2: Rule *iii.b* : For the sake of brevity, the transition above represents all transitions, where *(i)* $\ell, \ell' \subseteq 2^\Pi$, such that $rl \in \ell$ and $dotted, ll \in \ell'$, or $ll \in \ell$ and $dotted, rl \in \ell$, and *(ii)* $\ell, \ell' \subseteq 2^\Pi$, such that $rl \in \ell$ and $solid, ll \in \ell'$, or $ll \in \ell$ and $solid, rl \in \ell$.

While it is quite natural to disobey the direction and the soft lane change rules, a solid line should not be crossed. 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. Simulation Experiments

MVRRT* was implemented in C++ on a 2.2GHz processor with 4GB of RAM for the experiments in this section. We present a number of different scenarios in the same environment to be able to quantitatively compare the performance. In Fig. 4, the Dubins car 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, $\mathcal{S}_{rl}$ and $\mathcal{S}_{ll}$. A sidewalk $\mathcal{S}_{sw}$ 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 car 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 that 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.

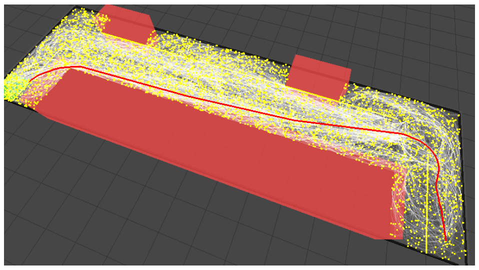
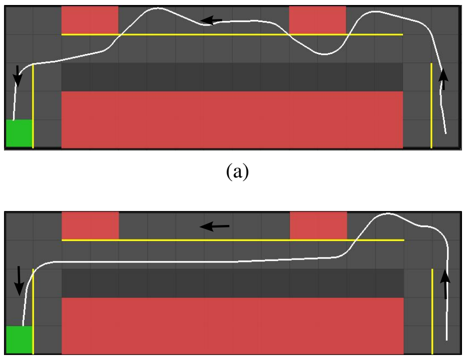


Fig. 3: MVRRT* tree after 40 sec. on an example without any safety rules. States of the Kripke structure are shown in yellow while edges are shown in white. The shortest trajectory shown in red to the goal region avoids obstacles but uses the sidewalk.



(a)

(b)

Fig. 4: Fig. 4a shows the solution after 60 secs. for 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.

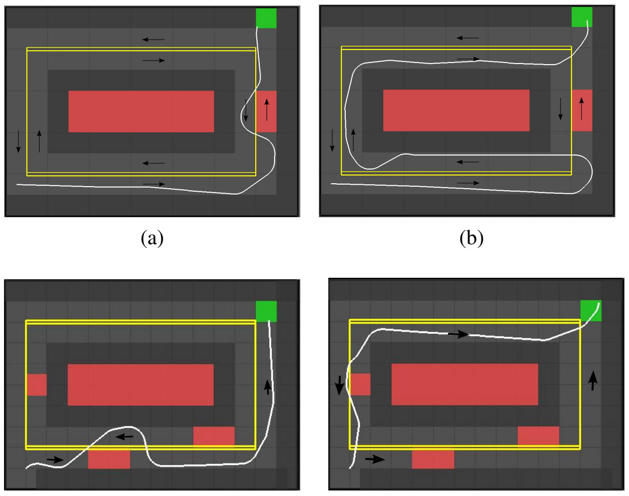
c) Case 3: Fig 5a shows a run for the sidewalk, direction and soft lane changing rules after 60 secs. of computation time with a level of unsafety of $(0, 0, 28.3)$. In Fig. 5b, with 120 secs. of computation, the solution has a much higher cost (215.8) but a significantly lower level of unsafety $(0, 0, 1.6)$ 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 incuring 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 $(0, 1, 25.17)$.

D. Implementation

In this section, we present results of our implementation of MVRRT* on an autonomous golfcart shown in Fig. 6 as a part of an urban mobility-on-demand system in the National University of Singapore's campus. The golfcart was instrumented with two SICK LMS200 laser range finders and has drive-by-wire capability. The algorithm was implemented inside the Robot Operating System (ROS) [17] framework.

Let us briefly describe the setup and note some major



(a) (b)

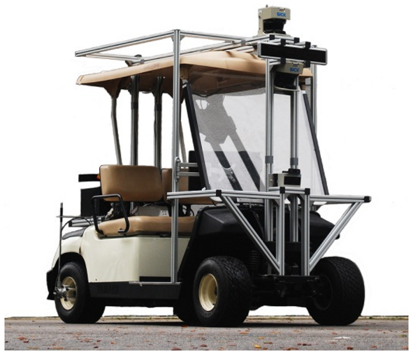
(c) (d)

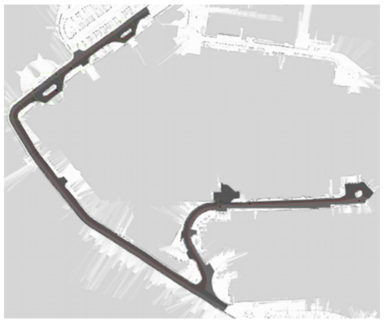
Fig. 5: Fig. 5a and 5b show the solution of MVRRT* after 60 and 120 secs. respectively, with the sidewalk, direction and soft lane changing rules. Note that the algorithm converges to a long trajectory which does not break any rules. Fig. 5c shows a solution after 20 secs. which breaks the hard lane 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.

implementation details. Traffic lanes and sidewalk regions are detected using pre-generated lane-maps of the campus roads, while obstacles are detected using data from laser range-finders. We use the sidewalk, direction and soft-lane changing rules for the experiments here. For an online implementation of MVRRT*, we incrementally prune parts of Kripke structure that are unreachable from the current state of the golfcart. The algorithm adds new states in every iteration (Lines 5-10 in Alg. 1) until the change in the level of unsafety of the best trajectory is within acceptable bounds between successive iterations. This trajectory is then passed to the controller that can track Dubins curves. We use techniques such as branch-and-bound and biased sampling to enable a fast real-time implementation and the golfcart can travel at a speed of approximately 10 kmph while executing the algorithm. Fig. 6 gives a snapshot of the experimental setup while Fig. 7 shows an instance of the golfcart going into the incoming lane in order to overtake a stalled car in its lane. Note that traffic in Singapore drives on the left hand side of the road.

VII. Conclusions

This paper considered the problem of synthesizing minimum-violation control strategies for continuous dynamical systems that obey a set of safety rules and satisfy a given reachability task. We focused on the case when the task is infeasible without breaking some of the safety rules. Ideas from sampling-based motion-planning algorithms and automata-based model checking approaches were utilized to propose an incremental algorithm to generate a trajectory of the dynamical system that systematically picks which safety rules to violate and minimizes the level of unsafety. The


(a)


(b)

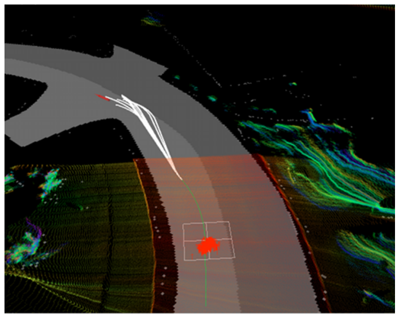
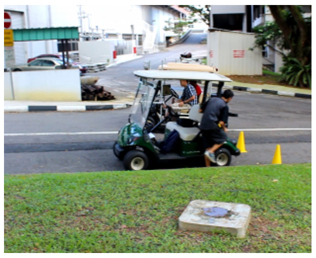
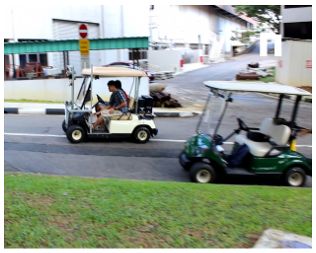

(c)

Fig. 6: Fig. 6a shows the Yamaha golfcart instrumented with laser range-finders and cameras. Fig. 6c shows the online implementation of MVRRT* in ROS. Red particles depict the estimate of the current position of the golfcart using laser data (shown using colored points) and adaptive Monte-Carlo localization on a map of a part of the NUS campus shown in Fig. 6b. Trajectories of the dynamical system, that are a part of the Kripke structure are shown in white while the trajectory currently being tracked is shown in green.

(a)


(b)


(c)

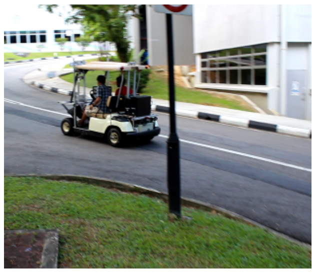

(d)

Fig. 7: The autonomous golfcart comes back into the correct lane after overtaking a stalled vehicle inspite of the road curving to the right. Note that the optimal trajectory without road-safety rules would cut through the incoming lane to reach the goal region.

algorithm was demonstrated in simulation experiments and also implemented on an experimental autonomous vehicle.

VIII. Ackowledgements

This work is supported in part by Michigan/AFRL Collaborative Center on Control Sciences AFOSR grant FA 8650-07-2-3744, US-NSF grant CNS-1016213, NSF-Singapore through FM SMART IRG, Nissan Motor Company and by the grant LH11065 at Masaryk University, Czech Republic.

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 Belta, 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.

[4] Stephen L Smith, Jana Tumova, Calin Belta, 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 Belta. 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.

[17] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: An open-source Robot Operating System. In *Workshop on Open-Source Software, ICRA*, 2009.