CYBER-PHYSICAL SYSTEMS

Memory-Loss Resilient Controller Design for Temporal Logic Constraints

M. Abate[a, b], W. Stuckey[c], L. Lerner[c], E. Feron[d], S. Coogan[a, e]

[a]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA; [b]School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA, USA; [c]Georgia Tech Research Institute, Atlanta, GA, USA; [d]Department of Electrical Engineering, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia; [e]School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, GA, USA

ARTICLE HISTORY
Compiled August 30, 2020

ABSTRACT
This paper studies the problem of controlling finite nondeterministic transition systems to satisfy constraints given as linear temporal logic properties. A controller architecture is proposed that maps finite fragments of the state trajectory history to control inputs. This approach avoids the standard controller construction that employs an onboard automaton which is fragile to memory loss or errors. In contrast, the proposed architecture requires storing only a finite sequence of previous system states in memory and is therefore resilient to memory loss. In particular, the system will operate unaltered after such a memory-loss event once the system recollects this finite sequence of system states. A generalized algorithm is outlined for controller synthesis in this manner. Additionally, we demonstrate the construction and implementation of such a memory-loss resilient controller through an experimental demonstration on a differential-drive robot that experiences memory-loss events.

KEYWORDS
Cyber-Physical Systems (CPS), Controller Synthesis, Resilience, Linear Temporal Logic

1. Introduction

When systems are required to satisfy complex temporal objectives (*e.g.*, specifications given in temporal logic), memoryless controllers are insufficient to ensure correct system behavior. A traditional approach for control subject to temporal logic constraints is to store relevant information about the system history in an appropriately constructed memory unit; runtime actions are then chosen as a function of the combined memory and system state [1–3]. This structure is employed, for instance, by controllers

CONTACT M. Abate. Email: Matt.Abate@GaTech.edu
CONTACT W. Stuckey. Email: William.Stuckey@GTRI.GaTech.edu
CONTACT L. Lerner. Email: Lee.Lerner@GTRI.GaTech.edu
CONTACT E. Feron. Email: Eric.Feron@Kaust.edu.sa
CONTACT S. Coogan. Email: Sam.Coogan@GaTech.edu

that rely on online automata [4–6]. Such controllers are convenient because they compactly represent relevant system history information into a finite number of memory states. However, they suffer from a need to maintain persistent knowledge over the entire operational life-cycle of the system; if at some time, the current automaton state is lost or altered, assurance cannot be provided going forward. For this reason, it is necessary to design controllers that operate without persistently stored information. Moreover, it is necessary to rectify the disconnect between controller designs that require persistent and precise knowledge of the system history with controller designs that allow for (if not rely on) regular software resets.

In instances where software aging degrades system performance, regular software restarts can be employed to add system robustness [7–10]. This method, referred to in literature as *software rejuvenation*, increases system resiliency by regularly reinstalling mission objectives from a trusted mission planner [11]. For controlled dynamical systems, methods exist for enforcing set invariance [12] and tracking control objectives [13] in the presence of such memory losses. Methods in this paradigm do not exist, however, for enforcing more complex logical and temporal system objectives.

This paper explores the problem of designing controllers to enforce linear temporal logic (LTL) specifications with no persistently stored information, *i.e.* no onboard automaton. That way, if at some time the system loses memory, it can restore correct action in a small amount of time. We create such a memory-loss resilient controller through an offline computation in the product-space between the system statespace and the statespace of an automaton corresponding to the operating specification; this allows the system designer to classify the minimum amount of information that is necessary at runtime to assure the system. Synthesis in the product space is standard in formal methods literature [1,2]; it is important to note, however, that while our proposed methodology uses an automaton for synthesis, the automaton itself is not implemented in the resulting controller architecture.

This paper is structured as follows: In Section 3 we contrast two potential controllers designed to enforce LTL specifications; The problem statement of the work is presented in Section 3 and then discussed in Section 4; We solve the problem statement in Section 5, and a generalized procedure is presented for generating controllers with no persistently stored information; The findings of this work are demonstrated in a case study, presented in Section 6.

2. Preliminaries

We model systems as finite-state nondeterministic transition systems, as formalized in Definition 2.1.

Definition 2.1. A *system* is a tuple $T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L)$, where

- $\mathcal{X}$ is a finite set of states,
- $\mathcal{U}$ is a finite set of control inputs,
- $f : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$ is a nondeterministic transition relation,
- Σ is a set of labels, and
- $L : \mathcal{X} \to \Sigma$ is a labeling map.

Such a system could, for instance, be a finite-state abstraction for a set of continuous-time differential equations, or a hybrid dynamical system; see [1,14] for further details. For the remainder of this paper, we use the notation $L(x_1, \cdots, x_n)$ to denote the

string of labels $L(x_1, \cdots, x_n) := L(x_1) \cdots L(x_n)$, and we use the symbols Σ^* and Σ^ω to denote the sets of finite and infinite words over Σ, respectively.

We specify mission objectives in linear temporal logic (LTL). LTL combines Boolean logical connectives such as disjunction, conjunction, and implication with temporal operators that permit, for example, specifying that a system criterion *eventually* holds along a system's execution, is *always* true along a system execution or is true *until* some other condition becomes true. In this way, LTL enables concisely and precisely specifying a wide range of operating behaviors. For a comprehensive discussion on the semantics of LTL, we refer the reader to [15, Chapter 5]. Additionally, we refer the reader to [1, Chapter 9] for a discussion on controller synthesis against LTL specifications.

Formally LTL specifications are interpreted over infinite system executions. To that end, we introduce the semantics of LTL_3 (Definition 2.2) in order to assess the finite system runs against LTL properties [16].

Definition 2.2 (LTL_3 Semantics)**.** Let $w \in \Sigma^*$ denote a finite word. The *truth value* of an LTL_3 formula φ with respect to w, denoted $[w \models \varphi]$, is an element of $\mathbb{B}_3 = \{\top, \bot, ?\}$ defined as follows:

$$[w \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega \ : \ w\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega \ : \ w\sigma \not\models \varphi \\ ? & \text{otherwise.} \end{cases}$$

Equivalently, the truth value of φ with respect to w is *true* "$\top$" if w is a good prefix for φ, *false* "$\bot$" if w is a bad prefix for φ, and *inconclusive* "?" otherwise.

Finally, we introduce the automata-based monitoring procedure for LTL_3.

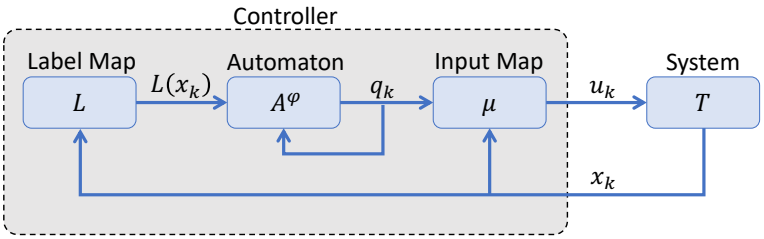
Definition 2.3 (Monitors in LTL_3)**.** For a given property φ a *monitor automaton* $\mathcal{M}^\varphi$ is a finite state machine that reads finite words $w \in \Sigma^*$ and outputs $[w \models \varphi]$.

For the remainder of this work, we use the tuple representation $\mathcal{M}^\varphi = (\Sigma, Q, q_0, \delta, \lambda)$ to denote the monitor automaton formed corresponding to φ, where

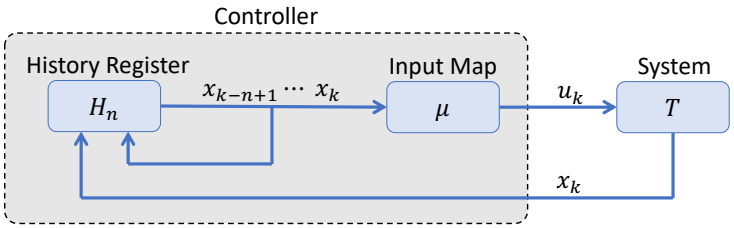
- Σ is a set of inputs,
- Q is a finite set of automaton states,
- $q_0 \in Q$ is an initial state,
- $\delta : Q \times \Sigma \to Q$ is a transition relation, and
- $\lambda : Q \to \mathbb{B}_3$ is a output function.

Additionally, for $q \in Q$ and $\sigma_1, \cdots, \sigma_n \in \Sigma$ we use the notation $\delta(q, \sigma_1 \cdots \sigma_n)$ to denote the state that q transitions to after receiving the string of labels $\sigma_1 \cdots \sigma_n$, defined recursively by $\delta(q, \sigma_1 \cdots \sigma_n) := \delta(\delta(q, \sigma_1 \cdots \sigma_{n-1}), \sigma_n)$.

Given an LTL property φ there may be many possible constructions of $\mathcal{M}^\varphi$. However, the authors of [16] show that each LTL property induces a unique minimal monitor automaton, that is, a monitor automaton $\mathcal{M}^\varphi$ with the fewest possible states. Additionally, the authors of [16] provide an algorithm for constructing this minimal monitor automaton directly from φ. For this reason, we assume hereafter that $\mathcal{M}^\varphi$ is always provided in its minimal form.



(a) Controller that assures T against φ. The system history at time k is stored in an automaton A^φ that receives the current system label $L(x_k)$ and the automaton state $q_k \in Q$. The control input u_k is chosen by the mapping $\mu : \mathcal{X} \times Q \to 2^{\mathcal{U}}$.



(b) Controller that assures T against φ. At time k, the n most recent system states $(x_{k-n+1}, \cdots, x_k) \in \mathcal{X}^n$ are stored in the history register H_n. The control input u_k is chosen by the mapping $\mu : \mathcal{X}^n \to 2^{\mathcal{U}}$.

Figure 1.: Two controller designs which assure the system $T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L)$ against the LTL specification φ.

3. Invertible Automata and Memory

Suppose a system designer would like to synthesize a controller that enforces the LTL property φ over a run of T. Intuitively, this controller will need access to information about the system history to assess the system's progress toward the satisfaction of φ and choose control inputs at runtime. We consider two possible controller architectures to assure the system T against φ. A block diagram visualization of each architecture is provided in Figure 1.

First, consider a controller that keeps track of the relevant information of the system history by running an onboard automaton A^φ (Figure 1a). For a detailed discussion on the generation and implementation of such controllers, we refer the reader to [1], where the control synthesis problem is solved using a game-theoretic approach. In this instance, the control input at time k is chosen by an input map $\mu : \mathcal{X} \times Q \to 2^{\mathcal{U}}$ that receives the current system state $x_k \in \mathcal{X}$ and the current automaton position $q_k \in Q$, where Q denotes the set of states of A^φ. This architecture is attractive because of its simplicity; knowledge of the current automaton state q_k is sufficient to assess the system's progress toward the satisfaction of φ. However, this design suffers from the fact that the automaton block must maintain an uncorrupted feedback loop for the entire operational lifetime of the system; if at some time, the controller loses track of its current position in A^φ, then all assurances going forward are lost.

As an alternative, consider a controller that chooses inputs by analyzing a finite fragment of the system's immediate history $x_{k+1-n} \cdots x_k$ (Figure 1b). We use the term *history register* to denote the internal component that stores such path fragments and the symbol H_n to denote a history register capable of storing n system states. Here, the control input at time k is chosen by an input map $\mu : \mathcal{X}^n \to 2^{\mathcal{U}}$

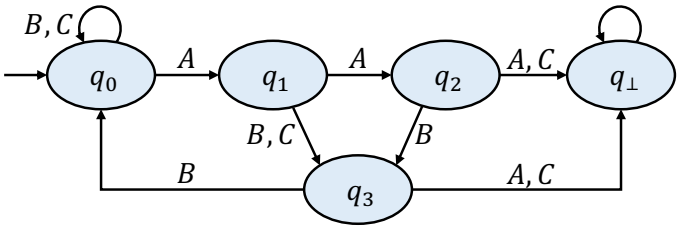


Figure 2.: Monitor automaton $\mathcal{M}^{\varphi_1}$ for the LTL specification $\varphi_1 = \square(A \rightarrow \bigcirc^2 B)$ evaluated over the alphabet $\Sigma = \{A, B, C\}$. States $q_0, \cdots, q_3$ have output *inconclusive*, " ? ", and state $q_\perp$ has output *false*, " $\perp$ ".

$L(x_{k-1})$	$L(x_k)$	If $[L(x_0 \cdots x_k) \models \varphi_1] \neq \perp$ then q_k equals
$(B \vee C)$	$(B \vee C)$	q_0
$(B \vee C)$	A	q_1
A	A	q_2
A	$(B \vee C)$	q_3

Figure 3.: Knowledge of the two previous system labels is always sufficient to correctly identify the system position inside $\mathcal{M}^{\varphi_1}$. This is exemplified by the fact that each potential two-state system history, *i.e.* combination of labels in Σ^2, corresponds to a unique current monitor position, provided that the previous system trace is known to be safe.

that receives the n most recent system states from H_n. A simple memoryless state feedback controller $u_k = \mu(x_k)$ can be thought of as a trivial instantiation of this proposed controller architecture, where $n = 1$, though we also allow for more general feedback control laws $u_k = \mu(x_{k-n+1}, \cdots, x_k)$. The addition of the history register to the controller architecture adds system resiliency by removing the need to maintain persistent information over the entire system life-cycle; if, at some time, the history register H_n loses track of the system history, then the system need only wait $n - 1$ time steps before the controller has the requisite information to continue choosing safe inputs.

Below we show that for certain LTL properties, knowledge of a finite history fragment is sufficient to correctly identify the system's progress toward the satisfaction of φ. This is the case, in particular, when the monitor automaton that corresponds to φ is *invertible*, *i.e.*, when there exists an $n \in \mathbb{N}_{\geq 1}$ such that knowledge of n previous system states is sufficient to identify the final state of the system run over $\mathcal{M}^{\varphi}$. We formally define monitor inversion in Definition 3.1 and we provide a sample construction of an invertible monitor automaton in Example 1.

Definition 3.1 (Invertible Monitor Automaton)**.** A monitor automaton $\mathcal{M}^{\varphi} = (\Sigma, Q, q_0, \delta, \lambda)$ is n-step invertible if, for all $w \in \Sigma^n$ and all $v_1, v_2 \in \Sigma^*$, $\delta(q_0, v_1 w) = \delta(q_0, v_2 w)$ whenever $\delta(q_0, v_1 w) \neq q_\perp$ and $\delta(q_0, v_2 w) \neq q_\perp$.

Example 1. Consider the LTL property $\varphi_1 = \square(A \rightarrow \bigcirc^2 B)$ evaluated over the alphabet $\Sigma = \{A, B, C\}$. We provide the corresponding monitor automaton $\mathcal{M}^{\varphi_1}$ in Figure 2. As enumerated in Figure 3, knowledge of the two previous system states is sufficient to identify the current monitor state, as each combination of labels in Σ^2

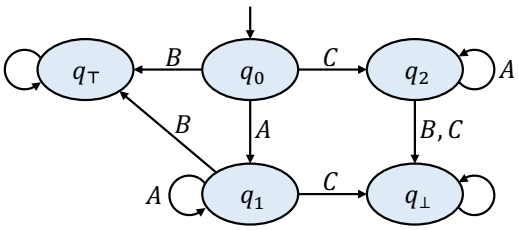


Figure 4.: Monitor automaton $\mathcal{M}^{\varphi_2} = (\Sigma, Q, q_0, \delta, \lambda)$ for the LTL property $\varphi_2 = \square(\bigcirc A) \vee (A\,U\,B)$ evaluated over the alphabet $\Sigma = \{A, B, C\}$. States q_0, q_1, and q_2 have output *inconclusive*, "?". States $q_\perp$ and $q_\top$ have outputs *false*, "$\perp$", and *true*, "$\top$", respectively.

corresponds to one and only one possible monitor position, provided that the previous system trace is known to be safe. Therefore, $\mathcal{M}^{\varphi_1}$ is n-step invertible for any $n \geq 2$. Further, it is not difficult to see that $\mathcal{M}^{\varphi_1}$ is not 1-step invertible, as knowledge of $L(x_k)$, the current system label, does not by itself uniquely determine the current monitor state. It follows, therefore that a history register based controller that stores two system states will be able to achieve the same assurances as a controller that runs $\mathcal{M}^{\varphi_1}$ onboard. This design, however, has added resiliency and is able to provide assurance in the presence of memory loss. □

Example 1 demonstrates that a history register style controller can provide assurance against φ by inverting the monitor automaton $\mathcal{M}^{\varphi}$ online and then synthesizing a safe control policy in the product space $\mathcal{X} \times Q$. Note, however, that not all automata are invertible. We demonstrate this assertion in Example 2.

Example 2. Consider the LTL property $\varphi_2 = \square(\bigcirc A) \vee (A\,U\,B)$ evaluated over the alphabet $\Sigma = \{A, B, C\}$. We provide the corresponding monitor automaton $\mathcal{M}^{\varphi_2}$ in Figure 4. Knowledge of a finite history fragment $x_{k-n+1} \cdots x_k$ will not always be sufficient to identify the system's progress toward satisfying φ_2. This is exemplified by the fact that if $L(x_{k-n+1} \cdots x_k) = (A)^n$, then it will not be possible to determine whether the system run over the monitor ends in state q_1, q_2 or $q_\top$; this is true regardless of the number of states that are stored. It follows, therefore that $\mathcal{M}^{\varphi_2}$ is not invertible. □

For this reason, it is favorable to abandon controller architectures that rely on onboard automata, and instead design controllers that map directly from the system state-space to a set of safe control inputs, with no intermediate automaton inversion. Synthesizing such a controller is equivalent to developing a mapping $\mu : \mathcal{X}^n \to 2^{\mathcal{U}}$ and a *safe* region of the label-space $S^{\mathrm{b}} \subseteq \mathcal{X}^n$ that is forward-invariant under μ. We formalize the search for such a mapping as the main problem statement of this work:

Problem Statement. Consider a system $T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L)$ and an LTL property φ. Given $n \in \mathbb{N}_{\geq 1}$, generate a feedback control policy $\mu : \mathcal{X}^n \to 2^{\mathcal{U}}$ and a region $S^{\mathrm{b}} \subseteq \mathcal{X}^n$ such that

(1) S^{b} is forward-invariant under μ, *i.e.* if $(x_1, \cdots, x_n) \in S^{\mathrm{b}}$ and $u \in \mu(x_1, \cdots, x_n)$, then $(x_2, \cdots, x_{n+1}) \in S^{\mathrm{b}}$, for all $x_{n+1} \in f(x_n, u)$, and
(2) S^{b} is *safe*, *i.e.* if $[L(x_0 \cdots x_k) \models \varphi] \neq \perp$, $(x_{k+1-n}, \cdots, x_k) \in S^{\mathrm{b}}$ and $u \in \mu(x_{k+1-n}, \cdots, x_k)$, then $[L(x_0 \cdots x_{k+1}) \models \varphi] \neq \perp$, for all $x_{k+1} \in f(x_k, u)$.

For the remainder of the work, we call $\mu_n : \mathcal{X}^n \to 2^{\mathcal{U}}$ an *n-step feedback control policy* if there exists an $S_n^{\mathrm{b}} \subseteq \mathcal{X}^n$ such that the pair $(\mu_n, S_n^{\mathrm{b}})$ solves the problem statement.

4. History Register Based Controllers for Safety

Note that employing an n-step feedback control policy μ_n does not guarantee that the resultant infinite system trajectory satisfies φ; rather, employing μ_n guarantees that at any given time $k \geq n$ the current finite system trajectory $[L(x_0, \cdots, x_k) \models \varphi] \neq \bot$. In this sense, one can view μ_n as a least-restrictive backup control policy that retains nondeterminism [17]. For certain classes of specifications, however, choosing control inputs according to μ_n guarantees infinite-time system satisfaction of φ; in this instance, μ_n can be employed stand alone. This observation is formalized in the next remark.

Remark 1. If φ is an LTL *safety* property and $(\mu_n, S_n^{\mathrm{b}})$ solves the problem statement, then the infinite system trace resulting when μ_n is applied satisfies φ, *i.e.* $L(x_0, x_1, \cdots) \models \varphi$. This is due to the fact that the satisfaction of a safety property φ, can be shown by proving the system trace does not contain a bad prefix for φ [16].

Additionally, note that the assurance capabilities of an n-step feedback control policy increase as n increases. This is due to the fact that the ambiguity as to the system's progress toward the satisfaction of φ is reduced as the number of saved system states increases. In the instance that the monitor automaton $\mathcal{M}^{\varphi}$ is invertible, however, there exists an intrinsic maximum number of states that will need to be remembered by a controller before there is no ambiguity as to the system's progress. We encapsulate this result in Theorem 4.1.

Theorem 4.1. *Let φ be an LTL property such that the monitor automaton $\mathcal{M}^{\varphi}$ is n-step invertible. Let the pair $(\mu_m, S_m^{\mathrm{b}})$ solve the problem statement where μ_m is an m-state feedback control policy $m > n$. Then there exists an n-step feedback control policy μ_n with a corresponding region S_n^{b}, such that if $(x_{k-m+1}, \cdots, x_k) \in S_m^{\mathrm{b}}$ then*

- $(x_{k-n+1}, \cdots, x_k) \in S_n^{\mathrm{b}}$, *and*
- $\mu_m(x_{k-m+1}, \cdots, x_k) = \mu_n(x_{k-n+1}, \cdots, x_k)$.

Theorem 4.1 follows immediately from the preceding discussion, and we sketch the proof for this result as follows: if the system remembers n previous states, then the monitor automaton $\mathcal{M}^{\varphi}$ can be inverted, to give μ_n access to the current state of the system execution over $\mathcal{M}^{\varphi}$. Storing additional system history states does not give the controller access to more information about the system's progress.

As shown in Theorem 4.1, the guarantees of a history-register based controller derive from the invertibility of the monitor automaton $\mathcal{M}^{\varphi}$. Intuitively, a non-invertible monitor automaton will contain two or more symmetrically labeled cycles, so that the system's progress toward the satisfaction of φ cannot be determined by any history fragment of finite length (see Example 2). As such, one can show that $\mathcal{M}^{\varphi}$ is n-step invertible by calculating all transition-cycles of length n and then checking that no two cycles are symmetric under cyclic permutation. This procedure can be accomplished using the standard graph search algorithms; transition-cycles, for instance, can be identified using a recursive *depth first search* to produce a modular decomposition of

all possible cycles of states in $\mathcal{M}^\varphi$. Such a search will produce a finite set of minimum length sub-cycles, from which all of possible cycles can be characterised.

Importantly, one must only calculate transition-cycles up to length $|Q|$ before invertibility can be determined. Therefore, the procedure for determining monitor invertibility has an inherent finite stopping time, and this time scales with the complexity of the specification. We formalise this result though the following proposition.

Proposition 4.2. *If the monitor automaton $\mathcal{M}^\varphi = (\Sigma, Q, q_0, \delta, \lambda)$ is invertible, then there exists an $n \leq |Q|$ such that $\mathcal{M}^\varphi$ is n-step invertible.*

Proof. For an LTL property φ, assume the monitor automaton $\mathcal{M}^\varphi$ is invertible. Choose $\sigma \in \Sigma$. As a consequence of the fact that $\mathcal{M}^\varphi$ is invertible, there cannot be a 2-state cycle in $\mathcal{M}^\varphi$ with edges labeled σ, *i.e.* there cannot be two states $q_1, q_2 \in Q$ such that $\delta(q_1, \sigma) = q_2$ and $\delta(q_2, \sigma) = q_1$. This assertion extends to cycles of any length and, thus, there must exist a unique $q_\sigma \in Q$ (possibly with $q_\sigma = q_\perp$) such that the following two conditions hold:

(1) $\delta(q_\sigma, \sigma) = q_\sigma$, and
(2) for all $q \in Q$ and $y \geq |Q|$ we have $\delta(q, \sigma^y) \in \{q_\sigma, q_\perp\}$.

Moving recursively, it can be shown that for every $w \in \Sigma^*$ such that $|w| \geq |Q|$ there exists a unique $q_w \in Q$ (possibly with $q_w = q_\perp$) such that

(1) $\delta(q_w, w) = q_w$, and
(2) for all $q \in Q$ we have $\delta(q, w) \in \{q_w, q_\perp\}$.

Therefore if $\mathcal{M}^\varphi$ is invertible, then $\mathcal{M}^\varphi$ is n-step invertible. This completes the proof. □

5. Computing an n-step feedback control policy

Consider a system $T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L)$. Given $n \geq 1$ and LTL specification φ, we aim to develop an n-step feedback control policy $\mu : \mathcal{X}^n \to \mathcal{U}$ that assures T against φ in some region $S^{\mathrm{b}} \subseteq \mathcal{X}^n$. Here, n could be chosen to either ensure the n-step invertibility of $\mathcal{M}^\varphi$, if $\mathcal{M}^\varphi$ is invertible, or to meet some other specification on the system design; in the latter case, the safe zone S^{b} may be smaller, however the system will recover faster in the event of memory loss. We solve the problem statement by applying the following steps:

(1) We first form a product system $P_n = T \otimes_n \mathcal{M}^\varphi$ which encodes the linked progression of the system and monitor automaton.
(2) We then define an equivalence relation $\sim$ that equates two product states if these states are both *safe* and observationally equivalent in a history register. We use the symbol $P_n/_\sim$ to denote the *quotient* of P_n under observational equivalence, which we formalize later in the work.
(3) Finally, we compute the largest forward-invariant region of the quotient transition system $S^{\mathrm{b}} \subset P_n/_\sim$, and a corresponding safe control policy $\mu : S^{\mathrm{b}} \to 2^{\mathcal{U}}$.

It is important to note that while our proposed methodology uses a monitor automaton to construct an n-step feedback control policy, the monitor is not used at runtime. Rather, control inputs are chosen online by the preformed policy μ.

In order to synthesize a controller capable assuring T against an LTL property φ, it is necessary to perform analyses in the product space of the system and the objective.

Definition 5.1. The n^{th} *controlled monitor product automaton* $P_n = T \otimes_n \mathcal{M}^\varphi$ of a system $T = \{\mathcal{X}, \mathcal{U}, f, \Sigma, L\}$ and a monitor automaton $\mathcal{M}^\varphi = \{\Sigma, Q, q_0, \delta, \lambda\}$ is defined to be $P_n = (\overline{\mathcal{X}}, \mathcal{U}, \overline{f}, \overline{O}, \overline{o})$, where

- $\overline{\mathcal{X}} \subseteq \mathcal{X}^n \times Q$ is a set of states, where

$$\overline{\mathcal{X}} = \Big\{ (x_{p+1}, \cdots x_{p+n}, q) \in \mathcal{X}^n \times Q \mid \exists x_0, \cdots, x_p \in \mathcal{X}, \exists u_0, \cdots, u_{p+n-1} \in \mathcal{U} \\ \textbf{such that } q = \delta(q_0, L(x_0, \cdots, x_{p+n})) \textbf{ and} \\ x_{i+1} \in f(x_i, u_i) \textbf{ for all } 0 \leq i \leq p+n-1 \Big\}$$

- $\mathcal{U}$ is a set of inputs,
- $\overline{f} : \overline{\mathcal{X}} \times \mathcal{U} \to 2^{\overline{\mathcal{X}}}$ is the transition map, where $(x_2, \cdots, x_{n+1}, q') \in \overline{f}(x_1, \cdots, x_n, q, u)$ if and only if $x_{n+1} \in f(x_n, u)$ and $q' = \delta(q, L(x_{n+1}))$,
- $\overline{O} = \mathcal{X}^n \cup \{x_\perp\}$ is a set of observations, and
- $\overline{o} : \overline{\mathcal{X}} \to \overline{O}$ is the observation map, where

$$\overline{o}(x_1, \cdots, x_n, q) = \begin{cases} x_\perp & \lambda(q) = \perp \\ (x_1, \cdots, x_n) & \text{otherwise.} \end{cases}$$

Each state of the product automaton $P_n = T \otimes_n \mathcal{M}^\varphi$ denotes a possible system history fragment paired with a possible monitor state; similarly, a run over the product automaton encodes a system trajectory and its corresponding monitor run. We therefore compute $\overline{\mathcal{X}}$ by first forming $\mathcal{X} \times Q$ and then removing the states that do not appear in valid system-monitor trajectories.

Note that the observation $\overline{o}(s)$ of a product state $s \in \overline{\mathcal{X}}$ is equal to $x_\perp$ if and only if the system trace up until this point is a bad prefix for the mission objective. Therefore system safety is only achievable in the instance that the system trace over the product automata $s_0, s_1, s_2, \cdots \in \overline{\mathcal{X}}^\omega$ does not contain a state with output false, *i.e.* $\overline{o}(s_i) \neq x_\perp$ for all i. For this reason, we allow the controller to choose control inputs based on the assumption that the system has acted safely up until the current time, *i.e.* at time k, the controller assumes $\overline{o}(s_k) \neq x_\perp$, where s_k denotes the unknown current system state in the product automaton. Under this assumption, the observation $\overline{o}(s_k)$ is equal to the total amount of information available to the controller, when the system and monitor is in state $s_k \in \overline{\mathcal{X}}$. Moreover, we can design controllers to assure the original system by developing policies that assure the product system given only the observation of the current state. We formalize this assertion as Proposition 5.2.

Proposition 5.2. *For* $\mu : \mathcal{X}^n \to 2^{\mathcal{U}}$ *and* $S^{\mathrm{b}} \subseteq \mathcal{X}^n$ *define*

$$\overline{S}^{\mathrm{b}} := \{ s \in \overline{\mathcal{X}} \mid \overline{o}(s) \in S^{\mathrm{b}} \}$$

and

$$N(s) := \bigcup_{u \in \mu(\overline{o}(s))} \overline{f}(s, u)$$

so that $\overline{S}^{\mathrm{b}} \subset \overline{\mathcal{X}}$ *is the set of product states whose observations are contained inside* S^{b}

and $N : \overline{\mathcal{X}} \to 2^{\overline{\mathcal{X}}}$ *is a function that takes a product state* $s \in \overline{\mathcal{X}}$ *and returns the set of product states reachable from* s *when the input is chosen by* μ*. If* $N(s) \subseteq \overline{S}^{\mathrm{b}}$ *for all* $s \in \overline{S}^{\mathrm{b}}$ *then the pair* (μ, S^{b}) *solves the problem statement.*

Proof. Assume $N(s) \subseteq \overline{S}^{\mathrm{b}}$ for all $s \in \overline{S}^{\mathrm{b}}$. Then $x_\perp \notin \overline{S}^{\mathrm{b}}$ and $\overline{S}^{\mathrm{b}}$ is forward invariant on P_n when control actions are chosen using μ. Further, S^{b} is forward invariant on T when control actions are chosen using μ, and all system runs constrained to S^{b} are not bad prefixes for φ. Therefore (μ, S^{b}) solves the problem statement. □

Next we equate the product states that are indistinguishable under observation. We refer to this procedure as taking the quotient of P_n under observational equivalence, as formalize in Definitions 5.3 and 5.4.

Definition 5.3. Let $P_n = (\overline{\mathcal{X}}, \mathcal{U}, \overline{f}, \overline{O}, \overline{o})$ be a product system. Two states $s_1, s_2 \in \overline{\mathcal{X}}$ are observationally equivalent (written $s_1 \sim s_2$) if and only if $\overline{o}(s_1) = \overline{o}(s_2)$.

Definition 5.4. Given a system $T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L)$ and a LTL property φ, let $P_n = (\overline{\mathcal{X}}, \mathcal{U}, \overline{f}, \overline{O}, \overline{o})$ be the product $P_n = T \otimes_n \mathcal{M}^\varphi$. The *automaton quotient* is defined to be $P_n/_\sim = (\mathcal{X}_\sim, x_\perp, \mathcal{U}, f_\sim)$, where

- $\mathcal{X}_\sim = \mathcal{X}^n \cup \{x_\perp\}$ is a set of states.
- $x_\perp$ is a *bad state*,
- $\mathcal{U}$ is a set of control inputs, and
- $f_\sim : \mathcal{X}_\sim \times \mathcal{U} \to 2^{\mathcal{X}_\sim}$ is a nondeterministic transition relation, $(x_2, \cdots, x_{n+1}) \in f_\sim(x_1, \cdots, x_n, u)$, if and only if there exists s_1, $s_2 \in \overline{\mathcal{X}}$ such that $s_2 \in \overline{f}(s_1, u)$, $(x_1, \cdots, x_n) = o(s_1)$, and $(x_2, \cdots, x_{n+1}) = o(s_2)$.

Recall that given a product state $s \in \overline{\mathcal{X}}$, the observation $\overline{o}(s)$ encodes the information that will be available to the controller in the instance that the system run over the product automaton ends in state s. As $x_\perp \in \overline{O}$ encodes the set of product states that are unsafe, system safety is ensured by choosing control inputs such that the system never enters $x_\perp \in \mathcal{X}_\sim$ in the quotient automaton. To that end, we strictly define the region from which assurance can be guaranteed; this region is referred to as a winning component of $P_n/_\sim$.

Definition 5.5. Let $P_n/_\sim = (\mathcal{X}_\sim, x_\perp, \mathcal{U}, f_\sim)$ be the quotient of $P_n = T \otimes_n \mathcal{M}^\varphi$ under product observational equivalence. A *winning component* $S^{\mathrm{b}} \subset \mathcal{X}_\sim$ is any set of quotient states such that for each history fragment $(x_1, \cdots, x_n) \in S^{\mathrm{b}}$ there exists an infinite sequence of inputs such that the system run over $P_n/_\sim$ starting from $(x_1, \cdots, x_n)$ never enters $x_\perp$. A *winning policy* is any mapping $\mu : \mathcal{X}_\sim \to 2^{\mathcal{U}}$ such that if $(x_1, \cdots, x_n) \in S^{\mathrm{b}}$, then $f_\sim(x_1, \cdots, x_n, u) \subseteq S^{\mathrm{b}}$, for all $u \in \mu(x_1, \cdots, x_n)$.

Numerous methods exist for identifying controlled invariant regions in automata [18,19]. Here, we include one such algorithm, specific to our application (Algorithm 1); $\textsc{Win}(P_n/_\sim)$ takes an arbitrary quotient system $P_n/_\sim$ and returns a maximal set of winning states $S^{\mathrm{b}} \subset \mathcal{X}_\sim$. The corresponding wining policy μ can then be calculated according to $\mu(x) = \{u \in \mathcal{U} \mid x \in S^{\mathrm{b}} \Rightarrow f_\sim(x, u) \subseteq S^{\mathrm{b}}\}$.

These tools are sufficient to solve the problem statement. Therefore, we now formalize the main result of the work (Theorem 5.6) and then demonstrate this result (Example 3).

Theorem 5.6. *Given a transition system,* T*, and an LTL property* φ*, let* $P_n/_\sim =$

Algorithm 1 Find S^{b} given $P_n/_\sim$

input : automaton quotient $P_n/_\sim = (\mathcal{X}_\sim, x_\perp, \mathcal{U}, f_\sim)$
output: largest winning component S^{b} of $P_n/_\sim$

function WIN($P_n/_\sim$)
  Initialize: $Bad \leftarrow \varnothing$, $Holder = \{x_\perp\}$.
  while $Bad \neq Holder$ **do**
    $Bad \leftarrow Holder$
    for $x \in \mathcal{X}_\sim \setminus Bad$ **do**
      if for all $u \in \mathcal{U}$ **there exists an**
      $x_b \in Bad$ **such that** $x_b \in f_\sim(x, u)$ **then**
        $Holder \leftarrow Holder \cup \{x\}$
  $S^{\mathrm{b}} := \mathcal{X}_\sim \setminus Bad$
  return S^{b}
end function

$(\mathcal{X}_\sim, x_\perp, \mathcal{U}, f_\sim)$ denote the quotient of $P_n = T \otimes_n \mathcal{M}^\varphi$ under product observational equivalence. Let $S^{\mathrm{b}} \subseteq \mathcal{X}_\sim$ be a winning component of $P_n/_\sim$, with a corresponding winning policy μ. Then the pair (μ, S^{b}) solves the Problem Statement.

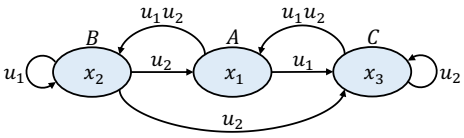
Proof. By definition, $S^{\mathrm{b}} \subseteq \mathcal{X}^n$ is forward-invariant under the control-policy μ. Consider a finite run of the system $x_0, \cdots, x_k$, such that $[L(x_0, \cdots, x_k) \models \varphi] \neq \perp$; by definition, the system run over the monitor automaton $\mathcal{M}^\varphi$ is guaranteed to end in a monitor state $q_k \neq q_\perp$. We can equivalently consider the system run over the product automaton $P_n = T \otimes_n \mathcal{M}^\varphi$, which will end in a state $s_k = (x_{k+1-n}, \cdots, x_k, q_k) \in \overline{\mathcal{X}}$ where $\overline{o}(s_k) = \tilde{s}_k \neq x_\perp$. Now consider the state $\tilde{s}_k \in \mathcal{X}_\sim$ in the automaton quotient $P_n/_\sim$, and assume $\tilde{s}_k \in S^{\mathrm{b}}$. If the control input $u_k \in \mu(\tilde{s}_k)$ is guaranteed to take $\tilde{s}_k$ to a state $\tilde{s}_{k+1} \in S^{\mathrm{b}}$, where $\tilde{s}_{k+1} \neq x_\perp$, then u_k is guaranteed to take s_k to a state $s_{k+1} = (x_{k+2-n}, \cdots, x_{k+1}, q_{k+1})$ such that $q_{k+1} \neq q_\perp$. Equivalently, $[L(x_0, \cdots, x_{k+1}) \models \varphi] \neq \perp$ for all $x_{k+1} \in f(x_k, u_k)$. Therefore, S^{b} is safe. □

Example 3. Consider a system whose dynamics are encoded in the transition system $T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L)$, where

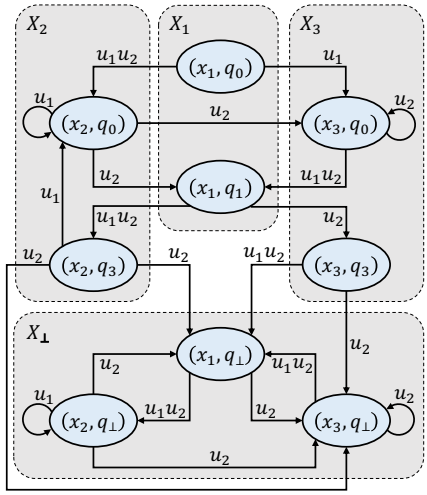
- $\mathcal{X} = \{x_1, x_2, x_3\}$, is a set of system states,
- $\mathcal{U} = \{u_1, u_2\}$ is a set of control inputs,
- $f : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$ is a transition relation, as given in Figure 5a,
- $\Sigma = \{A, B, C\}$ is a set of labels, and
- $L : \mathcal{X} \to \Sigma$ is a labeling map, as given in Figure 5a.

We aim to enforce the LTL property $\varphi = \square(A \to \bigcirc^2 B)$ (studied previously in Example 1) over a run of T, using a history-register style controller with $n = 1$ memorized system state. We synthesize such a controller by forming the product automaton $P_1 = T \otimes_1 \mathcal{M}^\varphi$ (Figure 5b), and then searching for the largest forward-invariant region S_1^{b} in the quotient automaton $P_1/_\sim$ (Figure 5c). In this case, assurance can be provided over $S_1^{\mathrm{b}} = \{x_1, x_2\} \subset \mathcal{X}$, where control inputs are chosen according to

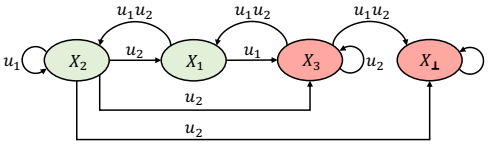
$$\mu_1(x) = \begin{cases} \{u_2\} & x = x_1 \\ \{u_1\} & x = x_2 \\ ? & x = x_3, \end{cases}$$



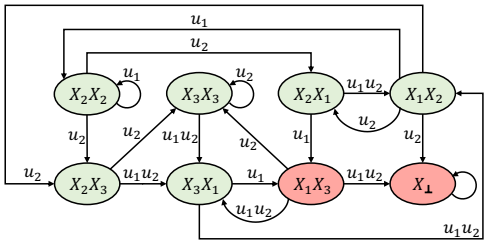
(a) Visualization of system dynamics for Example 3. The system is encoded in a three-state, nondeterministic transition system $T = (\mathcal{X}, \mathcal{U}, f, \Sigma, L)$.



(b) Product automaton $P_1 = T \otimes_1 \mathcal{M}^\varphi$. States which are observationally equivalent are grouped in grey, under the equivalence classes X_1, X_2, X_3 and $x_\perp$.



(c) Quotient system $P_1/_\sim$. The largest *winning* component is shown in light green, while the remaining *losing* states are shown in dark red.



(d) Quotient system $P_2/_\sim$. The largest winning component is shown in light green, while the remaining losing states are shown in dark red.

Figure 5.: Generating controllers with memory (Example 3). The quotient systems $P_1/_\sim$ and $P_2/_\sim$ are formed to classify a control policy that enforces $\varphi = \square(A \to \bigcirc^2 B)$ online when $n \in \{1, 2\}$ previous system states are known.

where the symbol "?" denotes the control output in the case that correct performance cannot be assured.

We now repeat this procedure to create the input map $\mu_2 : \mathcal{X}^2 \to 2^{\mathcal{U}}$. We synthesize this control policy by forming the two-state product automaton $P_2 = T \otimes_2 \mathcal{M}^\varphi$ and then searching for the largest forward-invariant region S_2^{b} in the automata quotient $P_2/_\sim$ (Figure 5d). In the case of this example, assurance can be provided over the region $S_2^{\mathrm{b}} \subset \mathcal{X}^2$ where control inputs are chosen according to

$$\mu_2(s) = \begin{cases} \{u_1\} & s = (x_1,\, x_2) \\ \{u_2\} & s \in \{(x_2,\, x_1),\, (x_3,\, x_3)\} \\ \{u_1, u_2\} & s \in \{(x_2,\, x_2),\, (x_2,\, x_3),\, (x_3,\, x_1)\} \\ \ ? & s = (x_1, x_3) \end{cases}$$

$$S_2^{\mathrm{b}} = \Big\{(x_1, x_2), (x_2, x_1), (x_3, x_3), (x_2, x_2), (x_2, x_3), (x_3, x_1)\Big\}.$$

Note that a controller that chooses inputs according to μ_2 will have significant advantages over a controller that chooses inputs according to μ_1. For instance, at a time k, a controller that chooses inputs according to μ_1 will be able to provide assurance if and only if the current system state $x_k \in \{x_1, x_2\}$; whereas, a controller that chooses inputs according to μ_2 will be able to provide assurance when $x_k = x_3$, provided that the previous system state is known to be $x_{k-1} \in \{x_2, x_3\}$. Additionally, in some cases μ_2 allows the system designer to choose which control input is applied. For instance, if, at a time k, the current system state $x_k = x_1$, then μ_1 will apply control input $u_k = u_2$ to T; whereas, in the same situation, the control policy μ_2 will allow the system designer to choose between the control input $u_k = u_1$ and $u_k = u_2$, provided that the previous system state is known to be equal to $x_{k-1} = x_3$. Finally, recall from Example 1 that $\mathcal{M}^{\varphi}$ is invertible with two memorized system. Therefore, increasing the memory capabilities of the controller beyond $n = 2$ memorized system states will not increase the number of instances when assurance is guaranteed, as shown in Theorem 4.1. □

6. Experimental Demonstration

We now present a case study demonstrating the findings of the work. In this study we design three different history-register based controllers to assure a differential drive robot against an LTL safety property. These controllers are generated using MATLAB 2020a and implemented on an actual robotic testbed.

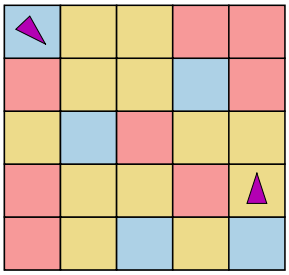
6.1. Controller Synthesis

Consider an autonomous vehicle which must refuel periodically, while also avoiding obstacles. We abstract the vehicle dynamics in a 200 state nondeterministic transition system T, representative of a discrete-time discrete-state unicycle model. Here, each state of T denotes a unique position $(x, y) \in \{1, \cdots, 5\} \times \{1, \cdots, 5\}$ and rotation $\theta \in \{0, \frac{\pi}{4}, \cdots, \frac{7\pi}{4}\}$. The control inputs to the system are linear velocity $v \in \{0, 1\}$ and angular velocity $\omega \in \{0, \pm\frac{\pi}{4}\}$. Nondeterminism in the system arises when the control input $(v, \omega) = (1, \pm\frac{\pi}{4})$, *i.e.* when the system intends to drive forward and turn during the same time-step. In this case the system will nondeterministically enter either the state arising from moving forward one unit and then rotating, or the state arising from rotating first and then moving forward one unit. This problem setting is shown in Figure 6.

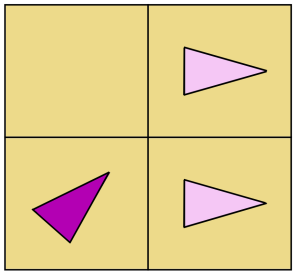
We stipulate that the vehicle must spend 2 consecutive time-steps in a refueling region every 8 time-steps and always avoid obstacles. This mission objective is captured in the LTL safety specification

$$\varphi = \Box(\neg Obs) \wedge \Box(\vee_{i=0}^{7} \bigcirc^i (Ref \wedge \bigcirc Ref))$$

where refueling states are labeled Ref and obstacle states are labeled Obs. We use the symbol $\varnothing$ to represent the system label when the vehicle is neither refueling or at an obstacle position. Therefore, we evaluate φ over the alphabet $\Sigma = \{Obs, Ref, \varnothing\}$; the labeling of T over Σ is shown in Figure 6a. The monitor automaton $\mathcal{M}^{\varphi}$ was



(a) Two-dimensional projection of system state space. Refueling (Ref) states are shown in blue, obstacle (Obs) states are shown in pink, and the remaining states ($\varnothing$) are shown in yellow. Vehicle states $x_1 = (1, 5, \frac{7\pi}{4})$ and $x_2 = (5, 2, \frac{\pi}{2})$ are shown as purple triangles, where $L(x_1) = Ref$ and $L(x_2) = \varnothing$.



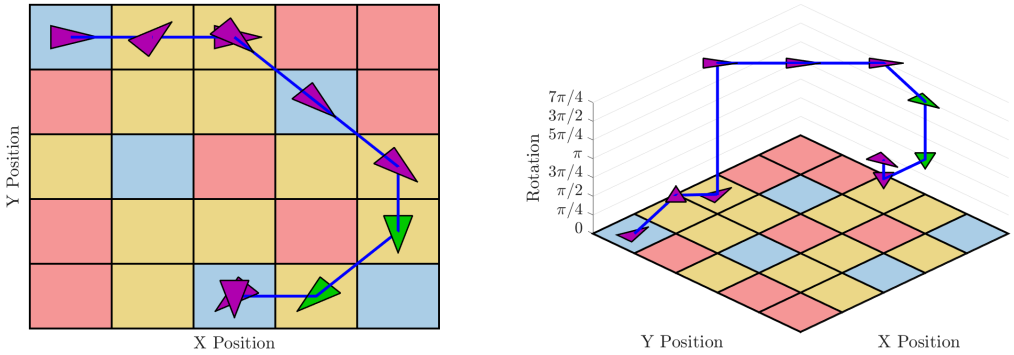
(b) Visualisation of nondeterminism in the system model. When beginning at an initial state $(1, 1, \frac{\pi}{4})$, shown in purple, and applying the control input $(v, \omega) = (1, -\frac{\pi}{4})$ the system will nondeterministically enter either of the states shown in light pink: $(2, 2, 0)$ which is the state arising from moving forward one unit and then rotating, or $(2, 1, 0)$ which is the state arising from rotating first and then moving forward one unit.

Figure 6.: Problem setting. Figure 6a shows the labeling of the system state space, and Figure 6b describes the nondeterminic control actions.

formed using Spot 2.0 [20]; the resultant automaton is composed of 16 monitor states, 15 of which output inconclusive and 1 of which is a false trap-state. A graph search conducted using MATLAB 2020a shows $\mathcal{M}^\varphi$ to be 7-step invertible; this search is conducted by computing all words $w \in \Sigma^7$ and then showing that $\delta(q_0, v_1 w) = \delta(q_0, v_2 w)$ all $v_1, v_2 \in \Sigma^*$ whenever $\delta(q_0, v_1 w) \neq q_\perp$ and $\delta(q_0, v_2 w) \neq q_\perp$. Therefore, a history register based controller that stores seven previous system states will provide the same guarantees as a traditional automaton based controller. In the remainder of this section, we explore the guarantees that are possible with even fewer memory states.

We now consider three alternative controllers constructed to assure T against φ where these controllers are required to make runtime decisions based on $n \in \{1, 2, 3\}$ memorized system states. These controllers are formed according to the procedures laid out in Section 5, *i.e.*, we identify a controlled invariant region of the quotient system $P_n/_\sim$ and the corresponding winning policy μ_n. In general, such controllers will not be able to guarantee φ from all initial states. Intuitively, this is because φ requires refueling only every 8 time-steps, but a controller with limited memory will need to act conservatively and guarantee refueling within its memory horizon as it cannot know of refueling events further in the past. However, if the system is initialized within the winning invariant region, then they controller will ensure that φ is satisfied along the infinite trace of the system execution.

First, a control policy $\mu_1 : \mathcal{X} \to 2^{\mathcal{U}}$ is formed to provide assurance when $n = 1$ system state is available to the controller. Note μ_1 is *memory-less* and makes control decisions based solely on the current vehicle state. μ_1 has a corresponding region $S_1^{\mathrm{b}} \subset \mathcal{X}$ and is able to provide assurance over $|S_1^{\mathrm{b}}| = 62$ of the 200 system states. Here, S_1^{b} is comprised of every refueling state, as well as the states that can reach a refueling state in one time step; μ_1 then either brings the vehicle to the refueling region or forces the vehicle to remain inside the fueling region. At any other state outside S_1^{b}, a memoryless controller is unable to guarantee satisfaction of φ since it is unable to reach a refueling state in the next step and has no memory of the last refueling event.



(a) Projection of system trajectory to the x–y plane. The system state is notated by a green triangle when a 3 state fragment of the system's history is not accessible.

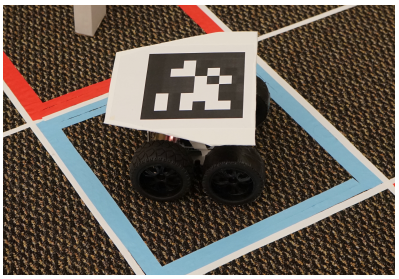
(b) Three-dimensional visualisation of system trajectory.

Figure 7.: Simulated system trajectory when μ_3 is employed. A 10 step simulation is conducted where the system begins at state $x_1 = (1, 5, 0)$ at time 1 and $L(x_1) = Ref$. A memory loss event occurs at time step 7 when the system is at state $x_7 = (5, 2, \frac{3\pi}{2})$ shown in green, and $L(x_7) = \varnothing$. By making intermediate decisions to avoid obstacles, the system is able to regain a full history register at time 9 and the system acts unaltered for the remainder of the study. Moreover, the resulting finite system trace over Σ before the history is recovered is not a bad prefix for φ and therefore, after recovery of the three-state system history, satisfaction of φ remains guaranteed.

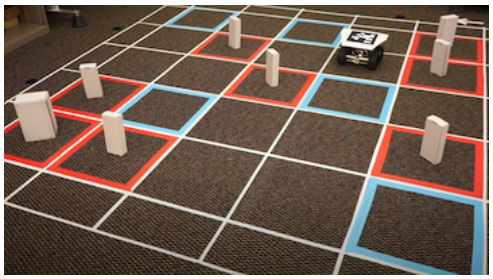
Next, a control policy $\mu_2 : \mathcal{X}^2 \to 2^{\mathcal{U}}$ is formed to provide assurance when $n = 2$ system states are available to the controller. μ_2 has corresponding region $S_2^{\mathrm{b}} \subset \mathcal{X}^2$, where assurance can be provided over $|S_2^{\mathrm{b}}| = 369$ possible two-state system histories. The controller μ_2 allows the vehicle to enter 123 of the possible 200 states without violating the system specification.

Finally, a control policy $\mu_3 : \mathcal{X}^3 \to 2^{\mathcal{U}}$ is formed to provide assurance with $n = 3$ stored history states. The controller μ_3 has a corresponding safe region $S_3^{\mathrm{b}} \subset \mathcal{X}^3$ and is able to provide assurance over $|S_3^{\mathrm{b}}| = 2310$ possible three-state system histories. This controller allows the vehicle to enter 134 of the possible 200 states without violating the system specification. Note that 64 of the 200 system states have label *Obs* and are inherently inadmissible; therefore, μ_3 allows the system to enter all but 2 admissible system states and is therefore almost as effective as the least conservative policy that requires seven history states. Provided that the initial state of the system and its history register is within S_3^{b}, the controller μ_3 ensures satisfaction of φ. Additionally, for certain three-state system histories, μ_3 allows the nondeterministic control input $(v, \omega) = (1, \pm\frac{\pi}{4})$, which is not the case for μ_1 and μ_2. This indicates that the controller has an assurance plan for both possible states at the next time-step and can allow the nondeterministic action without risk.

A simulated system trajectory under μ_3 is shown in Figure 7; this simulation is conducted in MATLAB 2020a. In the following section, we provide an experimental demonstration of μ_3 on a robotic testbed.



(a) PiBorg differential drive robot, shown refueling.



(b) Experimental setup. Refueling (Ref) states are shown in blue, and obstacle (Obs) states are shown in red.

Figure 8.: Robotic implementation. Figures 8a and 8b show experimental setup and the robotic platform.

6.2. Implementing a Memory-Loss Resilient Controller on a Differential Drive Robot

The three-state history register based controller μ_3 is implemented on a PiBorg model MonsterBorg car (Figure 8). The robot conforms to differential drive dynamics, and is controlled onboard by a Raspberry Pi. Localization is performed using APRILTAGS_ROS [21].

During periods of memory-loss, there is an inevitable risk that the system will falsify φ, *i.e.*, a bad prefix for φ may be encountered. To address this risk, we choose control actions during memory-loss according to the following principles: when the controller experiences a memory loss event, only the current system state will be available and, thus, μ_3 is no longer applicable for the next 2 timesteps. At this point, if the current system state is in S_1^{b}, then μ_1 is applied to the system for the next 2 timesteps and, in this case, system safety guaranteed for all future time. If instead the current system state is not contained in S_1^{b}, then the system attempts to steer to S_2^{b}; there is no guarantee in this procedure that the system will not violate φ at this timestep, however, if the system transitions to S_2^{b} without-reaching a bad prefix for φ then μ_2 can be applied at the next timestep and system safety is guaranteed for all future time. If the system cannot steer to S_2^{b}, then the system must steer to S_3^{b}; note that there will always be a control scheme which returns the system to S_3^{b} in 2 timesteps as S_3^{b} is forward invariant under μ_3, however, again there is no guarantee that φ will not be violated along the way. Once the system has regained a full history register, if a bad prefix for φ is not encountered, then the system regains its guarantee of satisfying φ for all future time.

In the experiment, the vehicle operated continuously for 3 minutes, and was subjected to two artificial memory-loss events. The resulting finite system trace over Σ is not a bad prefix for φ and, thus, the system safely recovered from the memory loss. A video of this experiment is available at https://www.youtube.com/watch?v=bn9fCWTrfpY.

7. Acknowledgements

The authors wish to thank Elizabeth Prucka for her work with testbed development.

8. Conclusion

This paper proposes a controller architecture that analyzes a finite fragment of the system history at each timestep in order to enforce a linear temporal logic property online. Unlike standard methods, the resulting controller does not rely on an onboard automaton, and therefore can withstand memory loss without compromising system safety. We demonstrate the implementation of such a memory-loss resilient controller through an experimental demonstration on a differential-drive robot.

References

[1] Belta C, Yordanov B, Gol E. Formal methods for discrete-time dynamical systems. Vol. 89. Springer; 2017.

[2] Yordanov B, Tumova J, Cerna I, et al. Temporal logic control of discrete-time piecewise affine systems. IEEE Transactions on Automatic Control. 2012 June;57(6):1491–1504.

[3] Liu J, Ozay N, Topcu U, et al. Synthesis of reactive switching protocols from temporal logic specifications. IEEE Transactions on Automatic Control. 2013 July;58(7):1771–1785.

[4] Kloetzer M, Belta C. A fully automated framework for control of linear systems from temporal logic specifications. IEEE Transactions on Automatic Control. 2008 Feb;53(1):287–297.

[5] Tabuada P, Pappas GJ. Linear time logic control of discrete-time linear systems. IEEE Transactions on Automatic Control. 2006 Dec;51(12):1862–1877.

[6] Loizou SG, Kyriakopoulos KJ. Automatic synthesis of multi-agent motion tasks based on LTL specifications. In: 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601); Vol. 1; Dec; 2004. p. 153–158 Vol.1.

[7] Jagtap P, Abdi F, Rungger M, et al. Software fault tolerance for cyber-physical systems via full system restart. CoRR. 2018;abs/1812.03546. Available from: http://arxiv.org/abs/1812.03546.

[8] Jiang L, Peng X, Xu G. Time and prediction based software rejuvenation policy. In: 2010 Second International Conference on Information Technology and Computer Science; July; 2010. p. 114–7.

[9] Garg S, van Moorsel A, Vaidyanathan K, et al. A methodology for detection and estimation of software aging. In: Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257); Nov; 1998. p. 283–292.

[10] Vaidyanathan K, Trivedi KS. A comprehensive model for software rejuvenation. IEEE Transactions on Dependable and Secure Computing. 2005 April;2(2):124–137.

[11] Huang Y, Kintala C, Kolettis N, et al. Software rejuvenation: analysis, module and applications. In: Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers; June; 1995. p. 381–390.

[12] Romagnoli R, Krogh BH, Sinopoli B. Design of software rejuvenation for CPS security using invariant sets. CoRR. 2018;abs/1810.10484. Available from: http://arxiv.org/abs/1810.10484.

[13] Romagnoli R, Krogh BH, de Niz D, et al. Software rejuvenation for secure tracking control. CoRR. 2018;Available from: http://arxiv.org/abs/1810.10468.

[14] Tabuada P. Verification and control of hybrid systems: A symbolic approach. 1st ed. Springer Publishing Company, Incorporated; 2009.

[15] Baier C, Katoen JP. Principles of model checking (representation and mind series). The MIT Press; 2008.

[16] Bauer A, Leucker M, Schallhart C. Runtime verification for LTL and TLTL. ACM Trans Softw Eng Methodol. 2011 Sep;20(4):14:1–14:64.

[17] Germn Rivera J, Andrs Danylyszyn A. Formalizing the uni-processor simplex architecture. 1996 02;.

[18] Lygeros J, Johansson KH, Simic SN, et al. Dynamical properties of hybrid automata. IEEE Transactions on Automatic Control. 2003 Jan;48(1):2–17.
[19] Tomlin CJ, Mitchell I, Bayen AM, et al. Computational techniques for the verification of hybrid systems. Proceedings of the IEEE. 2003 July;91(7):986–1001.
[20] Duret-Lutz A, Lewkowicz A, Fauchille A, et al. Spot 2.0 – a framework for LTL and ω-automata manipulation. In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis; (Lecture Notes in Computer Science; Vol. 9938); Oct. Springer; 2016. p. 122–129.
[21] Wills M. apriltag_ros wiki ; 2018. Available from: http://wiki.ros.org/apriltags_ros.