An MILP Approach for Real-time Optimal Controller Synthesis
with Metric Temporal Logic Specifications

Sayan Saha and A. Agung Julius

Abstract

The fundamental idea of this work is to synthesize reactive controllers such that closed-loop execution trajectories of the system satisfy desired specifications that ensure correct system behaviors, while optimizing a desired performance criteria. In our approach, the correctness of a system's behavior can be defined according to the system's relation to the environment, for example, the output trajectories of the system terminate in a goal set without entering an unsafe set. Using Metric Temporal Logic (MTL) specifications we can further capture complex system behaviors and timing requirements, such as the output trajectories must pass through a number of way-points within a certain time frame before terminating in the goal set. Given a Mixed Logical Dynamical (MLD) system and system specifications in terms of MTL formula or simpler reach-avoid specifications, our goal is to find a closed-loop trajectory that satisfies the specifications, in non-deterministic environments. Using an MILP framework we search over the space of input signals to obtain such valid trajectories of the system, by adding constraints to satisfy the MTL formula only when necessary, to avoid the exponential complexity of solving MILP problems. We also present experimental results for planning a path for a mobile robot through a dynamically changing environment with a desired task specification.

Keywords: temporal logic, reactive controller, mixed integer linear programming.

I. INTRODUCTION

At a high level, we synthesize a controller, by optimizing a desired cost function that provides inputs to a system, such that the behavior of the output satisfies some given requirements, such as the system avoids some unsafe behaviors and eventually terminates at some desired safe conditions. In recent years, a lot of attention has been given to temporal logic constraints, which have been used extensively for expressing reach-avoid specifications, safety requirements, and sequencing of tasks to be performed. These allow the designer to specify time-dependent constraints; for example, we may require that some property will eventually hold, or that some property holds until some other property is true. Using temporal logics allows much greater expressivity in defining desired system behaviors than their non-temporal counterparts, but at the cost of additional difficulty in satisfying the constraints.

A common approach to synthesize controllers to satisfy Linear Temporal Logic (LTL) properties is to create a finite abstraction model of the dynamical system which can be then used to synthesize controllers using an automata-based approach [1], [2], [3], [4]. This approach, however, results in high computational complexity due to quantization of the finite abstraction model and the size of the automaton may also be exponential in the length of the specification. In [5] the authors focus on coarse abstractions of the state-space to alleviate the increasing complexity problems as state-space dimension increases and also synthesize controllers for satisfying reactive tasks. A fine abstraction model for systems with complex dynamics is presented in [6] to synthesize reactive controllers by planning paths for a finite horizon, at the cost of *completeness* of the approach. Path planning using iterative sampling-based approach, while optimizing a certain cost and guaranteeing temporal logic specifications are presented in [7], [8]. Recently, researchers have been using mixed integer-linear programming to solve an optimal control problem by encoding LTL specifications [9], [10], [11], Metric Temporal Logic (MTL) specifications [12], and Signal Temporal Logic (STL) specifications [13] as mixed integer-linear constraints on the optimization variables. Reactive controller synthesis satisfying temporal logic specifications using receding horizon control has been considered in [14], [15], [16].

In this paper we consider MTL specifications, which augment the temporal operators with a metric interval or time bounds over which the operator is required to hold [17], [18], [19]. We consider the task of determining an input signal for a Mixed Logical Dynamical (MLD) system such that the system's output satisfies a given MTL specification. We address this problem by casting it as a Mixed Integer Linear Program (MILP). This concept has been previously applied by [13], [16] by encoding an STL specification ϕ in terms of an MILP, where a variable is associated with each time step and predicate, indicating the degree by which that predicate is satisfied by the associated output trajectory point of the system. The temporal operators and logical operators in the given specification are then broken down into a set of boolean operators, each of which are assigned a boolean variable along with a set of constraints that guarantee the variable is true when the boolean operator is satisfied, and the boolean variable is false when the operator is not satisfied. In all, this formulation introduces $O(N \cdot |\phi|)$ boolean variables and constraints, where $|\phi|$ denotes the number of operators in the STL specification and N is the horizon length over which the input signal is to be determined. We expect that linearly increasing the length of the trajectory or length of the STL specification will cause the time-complexity to grow exponentially since solving a MILP is exponential in the number of binary decision variables in the worst-case scenario [20]. This can quickly render the MILP intractable even for the relatively

Both the authors are with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180, Email: `sahas3,julia2@rpi.edu`.

small problems. This paper is motivated to circumvent this issue, such that the controller synthesis problem can be solved in real-time and can be applied in practical applications of robot motion path planning with temporal specifications. We propose a method whereby we dynamically identify the critical time-points over the simulation horizon, where the system trajectory violates the given MTL specifications most and introduce boolean variables and constraints only for those critical time-points iteratively and resolve the problem till the system trajectory satisfies the specification. This leads to a much smaller MILP problem to be solved leading to a significant reduction in the time required to generate the input signal. This approach is very closely related to the formulation presented in [21] for generating trajectories to avoid obstacles. We show the effectiveness of our algorithm by running experiments on a *m3pi* robot to plan a trajectory through a dynamically changing environment in real-time, so as to not hit any obstacles and reach a desired location within a given time limit.

II. PRELIMINARIES

We consider discrete-time systems of the form

$$x_{k+1} = F(x_k, u_k), \tag{1}$$

where, $x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_c} \times \{0,1\}^{n_l}$ are the continuous and binary/logical states, and $u_k \in U \subseteq \mathbb{R}^{m_c} \times \{0,1\}^{m_l}$ are the continuous and binary/logical control inputs at the time indices $k = 0, 1, \ldots$. We denote the system trajectory at time index k under the control input $\mathbf{u}^k = \{u_1, u_2, \cdots, u_k\}$ starting from a given initial condition $x_0 \in \mathcal{X}$, by $x(\mathbf{u}^k) = \{x_0, x_1, \cdots, x_k\}$. This system model provides a set of constraints for the optimization procedure, such that at any time index k the resulting trajectory $x(\mathbf{u}^k)$ satisfies the system dynamics given in (1). These constraints can be easily formulated in terms of mixed integer-linear program if the system under consideration belongs to the class of mixed-logical dynamical systems [22], that includes linear hybrid systems, constrained or unconstrained linear systems, piece-wise affine systems. Differentially flat and feedback linearizable systems [23], [24] can also be considered if the temporal logic specifications are in terms of the flat and observable outputs respectively.

An MTL formula is a formal language, that can be used to express desired properties that a system must satisfy with certain timing requirements. We consider the temporal operators *eventually* ($\Diamond_{[\mathcal{I}]}$), *always* ($\Box_{[\mathcal{I}]}$) and *until* ($\mathcal{U}_{[\mathcal{I}]}$), and logical operators, such as, *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), and *implication* ($\rightarrow$), that can be used to combine *atomic propositions* to form the MTL formula. We associate a set $\mathcal{O}(p) \subseteq \mathcal{X}$ with each atomic proposition p, such that p is true at time index k if and only if $x_k \in \mathcal{O}(p)$.

For example, using MTL one can easily express a desired system behavior that "the system trajectory should never enter some unsafe set $\mathcal{O}(p_{\mathtt{Unsafe}})$ and terminate in some desired set $\mathcal{O}(p_{\mathtt{Goal}})$ within time t_3 to t_4 and should pass through the set $\mathcal{O}(p_{\mathcal{W}})$ within the time frame t_1 to t_2 and must stay in $\mathcal{O}(p_{\mathcal{W}})$ once the trajectory enters it for s units of time" as

$$\phi = \Box \neg p_{\mathtt{Unsafe}} \wedge \Diamond_{[t_3, t_4]} p_{\mathtt{Goal}} \wedge \Diamond_{[t_1, t_2]} \Box_{[0, s]} p_{\mathcal{W}}.$$

Definition 1. *We define a system trajectory that satisfies the dynamics given in (1) to be an (in)feasible trajectory if it (falsifies)satisfies the given MTL specification.*

The robustness measure, ρ_ϕ of a system trajectory defines how robustly a system trajectory satisfies or falsifies the given MTL specification. The measure takes positive values if the trajectory satisfies the specification, and negative values otherwise. Intuitively, the robustness degree of an (in)feasible trajectory $x(\mathbf{u}^k)$ is the largest distance that we can independently perturb the points along the trajectory and maintain (in)feasibility. This defines a tube around the original trajectory such that any trajectory within this tube is guaranteed to satisfy (or falsify) the specification.

This concept is demonstrated in Fig. 1. In this figure we consider the MTL specification

$$\phi' = \Box_{[t_0, t_2]} \neg p_{\mathcal{B}} \wedge \Box_{[T, T]} p_{\mathcal{A}},$$

which states that between times t_0 and t_2 the trajectory should avoid the set $\mathcal{B}$ and at time T the trajectory is in set $\mathcal{A}$. Two trajectories are shown:

$x(\mathbf{u}_1^k)$ an infeasible trajectory;
$x(\mathbf{u}_2^k)$ a feasible trajectory.

Note that the trajectory $x(\mathbf{u}_1^k)$ has a large negative robustness, denoted by $\rho_{\phi'}^1$ and determined by the distance between the trajectory at time t_0, *i.e.*, $x_{k_{t_0}}$ (k_t represents the time index corresponding to the time t), and the boundary of the predicate $\mathcal{B}$. We call this time t_0 the *critical time* and the predicate $\mathcal{B}$ the *critical predicate*. By moving the *critical point* $x_{k_{t_0}}$ on the trajectory by any amount greater than this distance, we can push it outside of $\mathcal{B}$ in order to satisfy the specification. Analogously, trajectory $x(\mathbf{u}_2^k)$ has a smaller positive robustness, denoted by $\rho_{\phi'}^2$ and determined by the distance between the trajectory at time t_1 and the boundary of the predicate $\mathcal{B}$. This is because the trajectory at t_1 is closest to falsifying the specification, and would need to be moved by at least this amount in order to do so.

The tool `TaLiRo` [25] can be used to calculate the robustness, critical time, and critical predicate for a trajectory and MTL specification. The latter two values will be used to determine which constraints to add at each iteration of our method, presented in Section III.
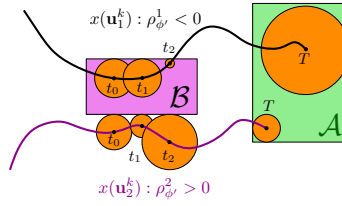
Figure 1. Illustration of robustness degree for MTL specification.

Finally, we assume that the set $\mathcal{O}(p)$ associated with each predicate p is polyhedral, defined by f faces. In this case, we can represent the set of points in the set uniquely by $\mathcal{O}(p) = \{x \mid Ax \leq b, \ A \in \mathbb{R}^{f \times n}, \ n = n_c + n_l, \ b \in \mathbb{R}^f\}$, with rows of A normalized to unit vectors. Further, we assume the MTL formula is expressed in negation normal form, which requires that the formula be transformed such that all negations, $\neg$, only appear immediately before a predicate. If we require predicate p to be true at time index k, that is, the trajectory at time index k has to be in $\mathcal{O}(p)$, then we can represent this via the constraints

$$Ax_k \leq b. \tag{2}$$

This requires the trajectory point x_k to lie in the intersection of all of the half-spaces of the faces of the predicate set $\mathcal{O}(p)$, which by definition is polyhedron.

Conversely, if we require predicate p to be false at time index k, that is, the trajectory at time k is not in $\mathcal{O}(p)$, then we can represent this via the constraints using the *big-M* formulation method by introducing a new binary decision vector z as

$$Ax_k + Mz \geq b, M \in \mathbb{R}_+, \ z \in \{0,1\}^f, \tag{3}$$

$$\sum_i z_i \leq f - 1, \tag{4}$$

where M is a value large enough to make the corresponding constraint hold for any allowable value of x_k. This requires the point x_k to lie outside of at least one half-space defining the polyhedron, which is both necessary and sufficient for the point x_k to lie outside of the polyhedron.

III. CONTROLLER SYNTHESIS

Problem 2. *Given an MLD system of the form (1), initial state $x_0 \in \mathcal{X}$, trajectory length N, correct system behavior defined in terms of an MTL specification ϕ, a desired robustness measure ρ_ϕ^d and a performance objective J, find*

$$\arg\min_{\mathbf{u}^N} \ J(x(\mathbf{u}^N), \mathbf{u}^N)$$

$$\textit{subject to } \rho_\phi(x(\mathbf{u}^N)) \geq \rho_\phi^d.$$

We propose a heuristic approach given in Algorithm 1 to solve the controller synthesis problem. The basic idea is to run the MILP first with only the system dynamic constraints (i.e. without MTL constraints). This formulation only contains boolean variables associated with the system dynamics in (1). If the MTL constraints are already satisfied, then we are done. Otherwise, we find the point that in some sense corresponds to the largest violation of the specification, and require this point to satisfy the corresponding predicate, and repeat. If the point is required to lie inside the polyhedron, then only the linear constraints (2) need to be added. If the point is required to lie outside of the polyhedron, then the f binary variables z and the linear constraints (3) need to be added. However, note that the solution of the MILP problem can satisfy the constraints given by (2) or (3) at equality and hence the resulting system trajectory though a feasible one, will have robustness measure $\rho_\phi(x(\mathbf{u}^N)) \geq 0$. Hence, in order to obtain a system trajectory with a desired robustness measure ρ_ϕ^d, we resize the predicate sets $\mathcal{O}(p)$ by ρ_ϕ^d. If the predicate p is such that it is a safe (unsafe) predicate, or in other words, the system can (never) visit the predicate, we shrink (bloat) the size of the predicate such that the new predicate set is given by

$$\tilde{\mathcal{O}}(p) = \{x \mid Ax \leq \tilde{b}\}, \ \tilde{b} = b \pm \rho_\phi^d \mathbf{1}_f, \tag{5}$$

where, $\mathbf{1}_f$ is a column vector of ones of size f. Identifying the safe and unsafe predicates can be done easily by parsing through the given MTL specification and marking the predicates with a negation ($\neg$) in front of them to be unsafe and the rest to be safe predicates.

The motivating idea behind this approach is that we do not require to put the constraints given by (2)-(3) at every time points of the trajectory; constraining only a few critical time points will result in the whole system trajectory to satisfy the MTL specifications in most cases. For instance (see Fig. 2), if we want the infeasible trajectory $x(\mathbf{u}_1^k)$ to satisfy the MTL specification ϕ' we first identify that the critical time and predicate are the time t_0 and $\mathcal{B}$ respectively by running TaLiRo as in Step 3 of Algorithm 1. The critical constraint corresponding to the time t_0 is then added to the MILP formulation, such that the trajectory point at t_0 is now constrained to lie outside the critical predicate $\mathcal{B}$ and the new MILP is solved as in

---

Algorithm 1 Open Loop Controller Synthesis

1: Initialize MILP with continuous state and input variables and constraints enforcing the linear dynamics in (1)

$$\text{MILP-cur} := \begin{cases} \underset{\mathbf{u}^N}{\arg\min} \ J(x(\mathbf{u}^N), \mathbf{u}^N) \\ \text{s.t. } x_{k+1} = F(x_k, u_k), \ x(\mathbf{u}^N) = \{x_i\}_{i=0}^N \end{cases}$$

2: Solve MILP-cur
3: Run `TaLiRo` on resulting trajectory to determine robustness, critical time, and critical predicate.
4: **while** robustness < 0 **do**
5:     **if** critical predicate is a safe predicate **then**
6:

$$\text{MILP-cur} := \begin{cases} \text{MILP-cur}, \\ \text{s.t. } Ax_k \leq b. \end{cases}$$

7:     **else**
8:

$$\text{MILP-cur} := \begin{cases} \text{MILP-cur} \\ \text{s.t. } Ax_k + Mz \geq b, \\ \quad \sum_i z_i \leq f - 1. \end{cases}$$

9:     **end if**
10:    Resolve MILP-cur
11:    **if** MILP-cur is an infeasible problem **then**
12:        Return "No feasible trajectory exists."
13:    **else**
14:        Rerun `TaLiRo`
15:    **end if**
16: **end while**
17: Return $x(\mathbf{u}^N), \mathbf{u}^N$

(a) Initial infeasible trajectory $x(\mathbf{u}_1^k)$: critical constraint corresponds to t_0.

(b) Infeasible solution trajectory $x(\mathbf{u}_{1'}^k)$ after solving the MILP first time: critical constraint corresponds to t_1.

(c) Feasible solution trajectory $x(\mathbf{u}_{1''}^k)$ after solving the MILP second time.
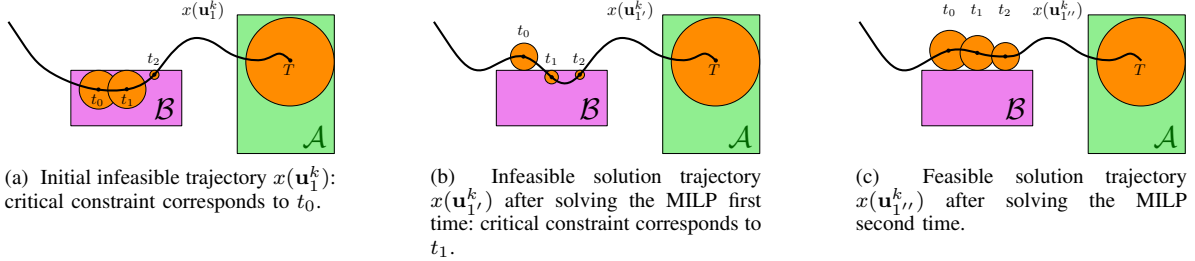
Figure 2.   Illustration of iterative addition of critical constraints in MILP formulation.

steps 5 and 6 of Algorithm 1. Assume, the solution of the MILP is $x(\mathbf{u}_{1'}^k)$. Since this resulting trajectory is still infeasible as found in Step 7, we repeat the process once again by adding constraints corresponding to the time t_1. After resolving the newer MILP, assume we obtain the trajectory $x(\mathbf{u}_{1''}^k)$, which turned out to be a feasible trajectory and hence the Algorithm 1 terminates. Note that we only added constraints for two time instances and do not need to introduce the additional binary variables associated with the trajectory point at time t_2. The goal is to reduce the computation time by iteratively solving much smaller MILPs, rather than the one large MILP.

However, iteratively adding critical constraints to the MILP problem in this way will result in an additional overhead in terms of encoding the MILP problem each time a new constraint is added. To circumvent this issue, we introduce a new scalar binary parameter $\sigma_{(i,j)}$ for each predicate i and each time index j and modify the linear constraints given in (2) and (3) for safe and unsafe predicates respectively as

$$Ax_k \sigma_{(\cdot,k)} \leq \tilde{b}\sigma_{(\cdot,k)},$$
$$(Ax_k + Mz)\sigma_{(\cdot,k)} \geq \tilde{b}\sigma_{(\cdot,k)}. \tag{6}$$

Note that if $\sigma_{(\cdot,k)} = 0$, then the constraint is trivially satisfied and hence relaxed. Only when $\sigma_{(\cdot,k)} = 1$ the constraint is required to hold. Using this modification, we encode the linear constraints for all the predicates at all the time indices $k = \{0, 1, \cdots, N\}$

based on whether the predicate is safe and unsafe right at the beginning. We also set all $\{\sigma_{(i,j)}\}$ to zero. Afterwards, as we proceed through the steps of Algorithm 1, we set one of $\{\sigma_{(i,j)}\}$ to 1 corresponding to the critical predicate and the critical time for that iteration. In this way, we add the new constraints to the MILP problem iteratively without having to encode the problem at each step.

In general, this method is not guaranteed to find a feasible solution if one exists, and is not guaranteed to return the optimal trajectory, even if a feasible solution is found. However for a fragment of MTL specifications we have the following result.

Theorem 3. *If the MTL specification consists of only conjunctions* ($\wedge$) *and the globally operator* ($\square$), *then if Algorithm 1 finds a feasible trajectory then it is the optimal trajectory. If Algorithm 1 fails to find any feasible trajectory, then Problem 2 is infeasible.*

Proof: A full-scale optimization approach for solving Problem 2 with such MTL specifications will require constraints given in either (2) or (3) to hold at all the time indices for all the predicates in the specification. Whereas, Algorithm 1 requires only a subset of those constraints to hold. Let us denote the search-spaces explored in full-scale optimization and in the final iteration of Algorithm 1 by U^* and U' respectively. Since, any feasible trajectory for the full-scale optimization problem is also a feasible trajectory for Algorithm 1 we have that, $U^* \subseteq U'$. Now, if Algorithm 1 returns a feasible solution optimal over U', it should also be a feasible solution in the U^* space, according to construction of Algorithm 1, implying it is also the optimal solution. However, if the MTL specification involves *disjunction* or other temporal operators that can be broken down in terms of *disjunction* (either this holds *or* that holds), then not all feasible trajectories of full-scale optimization problem are feasible for Algorithm 1. If some disjunction is the critical portion of the specification that defines the robustness, Algorithm 1 will require the trajectory to satisfy one particular predicate in the *disjunction*, however the full-scale optimization will require the trajectory to satisfy any of the predicates in the *disjunction* and hence $U^* \not\subseteq U'$. ∎

For our numerical examples, presented in Section V we consider the performance criteria

$$J = \sum_{k=1}^{N} \|\text{diag}(\mathbf{R})u_k\|_1 , \tag{7}$$

to minimize the control effort, where $\text{diag}(\mathbf{R})$ is a diagonal matrix consisting of the elements of the non-negative weighting row-vector $\mathbf{R}$. Introducing slack variables s_{kj}, with $k = 1, \ldots, N$, $j = 1, \ldots, m$, $m = m_c + m_l$, the objective function can be reformulated as

$$J = \sum_{k=1}^{N} \mathbf{R}s_k, \tag{8}$$

with an added set of constraints,

$$-s_{kj} \leq u_{kj} \leq s_{kj} \tag{9}$$

where, u_{kj} denotes the j-th component of the control input u_k at time index k, resulting in a linear optimization problem.

The trajectory length N depends on the *bound* of a *bounded-time* temporal specification (does not contain any unbounded operator), which is computed to be the "maximum over the sums of all nested upper bounds on temporal operators" [13].

IV. REACTIVE CONTROLLER SYNTHESIS

In this section, we present a receding horizon controller (RHC) framework to make the MILP controller presented in Section III, reactive to dynamically changing predicate sets. At each step $i = \{1, 2, \cdots, N\}$ of the RHC computation, we keep track of the system trajectory for time indices $\{0, 1, \cdots, (i-1)\}$ and search for a system trajectory of length $(N - i + 1)$, starting from the initial condition x_{i-1}, such that the combined system trajectory satisfies the given MTL specification with the desired robustness measure. While computing for the system trajectory at step 6 of the Algorithm 2, we use the open loop controller presented in the Algorithm 1, with one slight modification: searching for the critical time and critical predicate is based on the combined system trajectory $\{x_0, \cdots, x_{i-2}, x_{i-1}, \hat{x}_i, \cdots, \hat{x}_{N-i+1}\}$. Also since, each predicate set can be defined uniquely in terms of the (A, b) pair as given in (5), the MILP controller takes the (A, b) pair as arguments, instead of constant matrices, to account for any predicate sets that might change through the execution of the whole control plan.

As the end goal of this work is real-time controller synthesis, ideally it is desired that the steps 3 to 8 of the Algorithm 2 be executed within the time-step of the system dynamics, such that at each time-step the system has a control input signal to execute. However, this is usually not the case in the experiments we have performed. Step 6 is the rate limiting step of the process, since it involves iteratively solving an MILP problem if the predicate sets are dynamically changing and the upper bound to the number of iterations required to find a feasible system trajectory is $O(N \cdot |p|)$, where $|p|$ is the number of predicate sets in the MTL specification, in the worst-case scenario. But the incremental nature of Algorithm 1 allows us to execute the steps 3 to 8 only for the duration of the time-step of the system dynamics. Consider the example presented in Fig. 6, where instead of waiting for a feasible trajectory to be found in a single iteration (time for which may exceed the time-step), the controller iteratively finds a system trajectory that approaches towards a feasible one and requires 3 iterations to find a feasible

---

Algorithm 2 Receding Horizon Controller Synthesis

1: Run Algorithm 1 for trajectory length N.
2: Set initial condition to be: x_0.
3: **for** $i = \{1, \cdots, N+1\}$ **do**
4:     Set past system trajectory: $\{x_0, \cdots, x_{i-1}\}$.
5:     Compute the (A, b) pair defining the predicate sets.
6:     Obtain new system trajectory by running Algorithm 1 for trajectory length $(N - i + 1)$: $\{x_{i-1}, \hat{x}_i \cdots, \hat{x}_{N-i+1}\}$.
7:     Set initial condition for the next iteration to be: $\hat{x}_i$.
8: **end for**

---

trajectory after a new unsafe region was introduced in the workspace. Thus, obtaining a feasible trajectory with the desired robustness may take a few iterations, but the Algorithm 2 can essentially be run in real-time for this example.

V. EXAMPLES

A. Numerical Example

For numerical simulations, we consider two different MTL specifications for motion planning of a mobile robot with unicycle dynamics, and compare our results by running the same examples with the `BluSTL`[1] toolbox developed using the ideas in [13], [16]. All the simulations were performed on a computer with a 3.4 GHz *Intel core i7* processor with 16 GB of memory using `Gurobi`[2] solver through `YALMIP` [26].

We first consider a simple reach-avoid scenario (see Fig. 3), where the system has to avoid an unsafe area in its workspace at all times and reach the goal area and stay there from 8.5 to 10 seconds. These requirements can be represented using MTL specifications as,

$$\phi_1 = \left(\Box \neg p_{\texttt{Unsafe}}\right) \wedge \left(\Box_{[8.5,10]} \; p_{\texttt{Goal}}\right).$$

For the second example we consider both the *eventually* and *globally* operator in a nested fashion to show that our approach can handle complex task specifications as well. In this scenario the requirement is still to avoid the unsafe region always and to eventually reach the goal region sometime within 5.5 to 7.5 seconds and stay there for 1.5 seconds. Formally, the specification is

$$\phi_2 = \left(\Box \neg p_{\texttt{Unsafe}}\right) \wedge \Diamond_{[5.5,7.5]} \; \left(\Box_{[0,1.5]} \; p_{\texttt{Goal}}\right).$$

In order to implement our approach we first feedback linearize (see Section V-B) the unicycle dynamics to obtain double integrator dynamics in X and Y directions, governing the evolution of the position of the mobile robot. We then discretize the continuous system dynamics with a 0.5 second sample time, resulting in a trajectory length of $N = 20$ for the cases considered here.

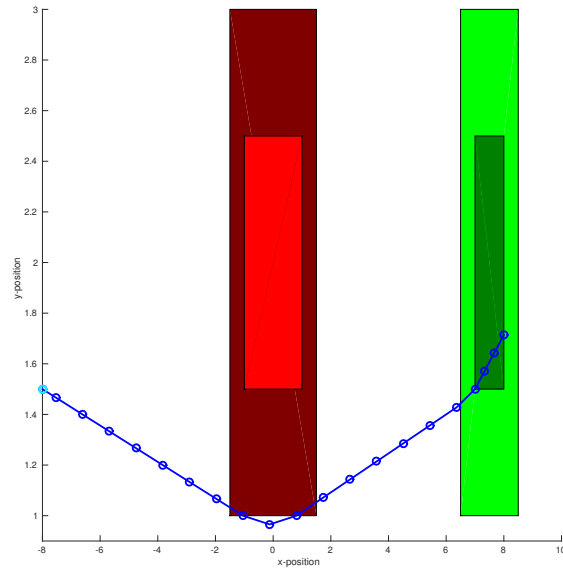[1]https://github.com/BluSTL/BluSTL/
[2]http://www.gurobi.com/

Figure 3. Illustration of path planning of mobile robot with the simple reach-avoid specification. The unsafe and the goal regions are colored red and green respectively. The unsafe set is bloated and the goal set is shrunk by the desired robustness degree, chosen to be 0.5 (trajectory points should not be inside the bloated unsafe set and outside of the shrunk goal set at the relevant time-instants). The system starts from the cyan colored position on the left side of the figure. Both BluSTL and our approach produce the same solution trajectory.

Method	MTL spec	YALMIP Time (s)	Open Loop Time (s)	RHC Time (s)
Our Approach	ϕ_1	3.12 ± 0.49	0.98 ± 0.13	10.02 ± 0.004
	ϕ_2	3.02 ± 0.78	0.84 ± 0.04	10.02 ± 0.001
BluSTL	ϕ_1	38.22 ± 2.16	5.83 ± 0.21	60.62 ± 1.06
	ϕ_2	39.21 ± 2.52	77.63 ± 2.15	1401.37 ± 19.50

Table I
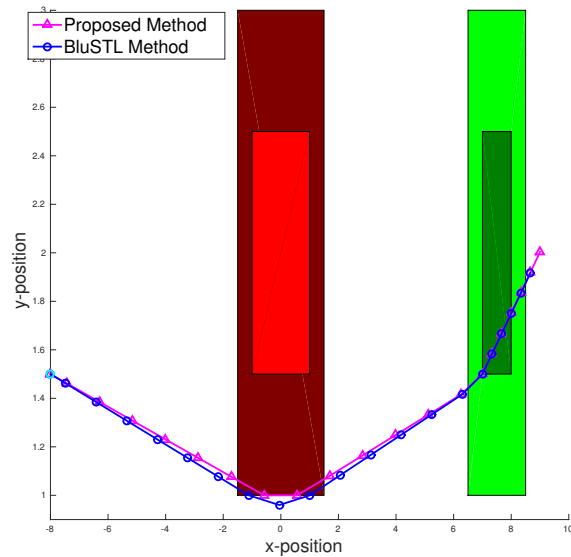TIME TAKEN TO SOLVE THE PATH PLANNING PROBLEM.

Figure 4. Path planning solutions of the mobile robot with complex task specifications. The workspace is same as before.

As expected, based on Theorem 3, both `BluSTL` and our approach produce the same optimal path for the specification ϕ_1 as shown in Fig. 3. However, for the task specification ϕ_2, because of the presence of the *eventually* operator in the specification `BluSTL` provides a more optimal path than our approach[3], even though both the solutions satisfy the specification ϕ_2 with the desired robustness as shown in Fig. 4. In Table I, we present the results obtained for the path planning problems in 'mean $\pm$ standard deviation' format obtained over 10 independent runs of the same problem. *YALMIP Time* represents the time taken to encode the controller and *Open Loop Time* and *RHC Time* are the times required to generate the feasible path in the open loop fashion and by using the receding horizon controller approach respectively using the `Gurobi` solver. Note that, the *RHC Time* represents the time taken for planning the path over $N = 20$ time-steps each of which is of duration 0.5 seconds. The timing results clearly shows that our approach is much faster in obtaining the solution trajectory as compared to `BluSTL`, specifically in the case of the more complex specification ϕ_2. As our end goal is to implement this controller synthesis procedure in a practical situation, being able to plan the path in a short amount of time is of a great importance.

B. Practical Example

Experimentally, we determine the efficacy of our MILP approach for reactive controller synthesis using a *m3pi* robot with a differential drive system dynamics. The *m3pi* robot consists of a *3pi* robot base connected to a *m3pi* expansion board that allows us to communicate with the robot using *XBee* wireless communication module to send control input signals from a workstation to the robot.

Denoting the position of the robot in a 2D plane to be (x, y), the equations of motion governing the system dynamics are

$$
\begin{aligned}
\dot{x} &= \frac{v_r + v_l}{2} \cos(\theta) = v \cos(\theta), \\
\dot{y} &= \frac{v_r + v_l}{2} \sin(\theta) = v \sin(\theta), \\
\dot{\theta} &= \frac{v_r - v_l}{2d} = \omega,
\end{aligned}
\tag{10}
$$

where, θ denotes the orientation of the robot with respect to the coordinate frame of reference and v_r and v_l are the wheel speeds of the right and left wheels respectively and are the control input signals to the robot. Linear and angular velocities of the robot are denoted by v and ω respectively for ease of notation and d is the distance of any one wheel from the center of the robot base. To design a controller for the unicycle agents we utilize a well-known theory from nonlinear control called feedback linearization [23] so that the nonlinear dynamics presented in (10) can be represented by second order particle dynamics in two dimensions. Choosing the position of the robot as the system output notice that,

$$
\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{v} \\ v\omega \end{bmatrix}.
$$

Then, by choosing the intermediate control inputs to the robot to be, $\dot{v}$ and $v\omega$ such that,

$$
\begin{bmatrix} \dot{v} \\ v\omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix},
\tag{11}
$$

where, u_x and u_y are the new control inputs to be determined, leads to the input-output feedback linearized system corresponding to second order particle dynamics in the form of a chain of integrators as,

$$
\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \end{bmatrix}.
\tag{12}
$$

These new control inputs u_x and u_y are then generated using the MILP approach presented in Section III. Using (11), one can derive the control inputs v and ω, and then the original control inputs

$$
v_r = v + d\omega \quad \text{and} \quad v_l = v - d\omega,
$$

so that the resulting system trajectory of the system satisfies the given MTL specification.

---

[3] Open loop and closed loop receding horizon implementation in the BluSTL toolbox actually produced different trajectories even with the exact same system parameters and specifications; closed loop trajectory obtained was non-optimal.
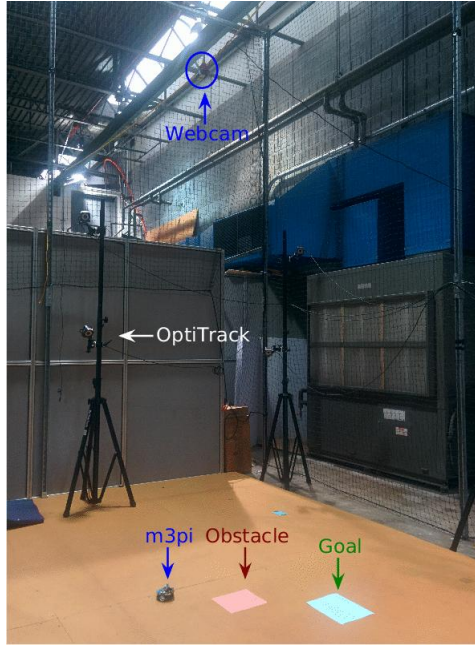
Figure 5. Experimental Setup for path planning of *m3pi* robot

The workspace environment for running the experiments is shown in Fig. 5. We use an overhead webcam to obtain images of the environment, which are then processed to determine the locations of the predicates, represented by the papers laid on the floor. The OptiTrack system is used to track the position of the *m3pi* robot accurately. A video of three different experiments, as well as an user-interactive example are uploaded here [4]. Here, we present a simulated version of one of the practical experiments in Fig. 6. The desired task specification is again a reach-avoid criteria

$$\phi_3 = \left(\Box \neg p_{\mathtt{Unsafe}_1}\right) \wedge \left(\Box_{[17.5,20]}\ p_{\mathtt{Goal}}\right),$$

where $\mathtt{Unsafe}_1$ is a previously known unsafe region in the workspace of the robot as shown in Fig. 6a. We assume that the workspace is changing dynamically such that as the robot is navigating through the workspace towards the goal region, it might come across another unsafe region all of a sudden. With this knowledge we encode our MILP controller with one safe predicate and two unsafe predicates as detailed in Section III. Since, we pass the (A, b) pair defining any predicate as parameters to the MILP controller we do not need to know the exact location of the $\mathtt{Unsafe}_2$ at the beginning of the path planning problem. When the system becomes aware of the new unsafe region it updates the corresponding (A, b) pair values and solves for a new feasible trajectory from its current position using Algorithm 2. As shown in Fig. 6, a previously unknown unsafe area pops up at 7.5 seconds and as can be seen from Fig. 6c - Fig. 6e the robot finds a feasible trajectory in 3 time-steps, due to the reason that we limit the duration of steps 3 to 8 of the Algorithm 2 to time-step of 0.5 seconds.

VI. CONCLUSION AND FUTURE WORK

In this paper, we consider the problem of optimizing the inputs to an MLD system such that it satisfies an MTL specification. We do this by finding the trajectory points that most violate the specification, constraining them to satisfy the corresponding predicate, and resolving the resulting MILP optimization problem. Although this problem can be fully formalized as an MILP optimization problem and solved directly, this introduces a number of binary variables and constraints that are linear in the length of the trajectory and size of the MTL specification. Our approach iteratively adds constraints, and require solving MILPs multiple times rather than once but can yield a low-cost feasible solution much faster by considering the smaller MILPs, instead of one large MILP.

We present the efficacy of our approach by finding feasible trajectories corresponding to two different MTL specifications by solving the optimization problem for a mobile robot in a few seconds. The numerical results show that this approach can generate feasible trajectories by adding only a few of the binary variables and constraints that would otherwise have been added. This motivates the current use versus the full MILP, as is usually done, which would have included hundreds of binary variables and constraints. We also show the reactiveness of the proposed approach by implementing the controller in real-time on a *m3pi* robot in a dynamically changing environment. Future work involves exploring heuristic approaches so as to use a linear combination of the time-points at which the MTL specification is violated to add the constraint, rather than just using the critical time-points.
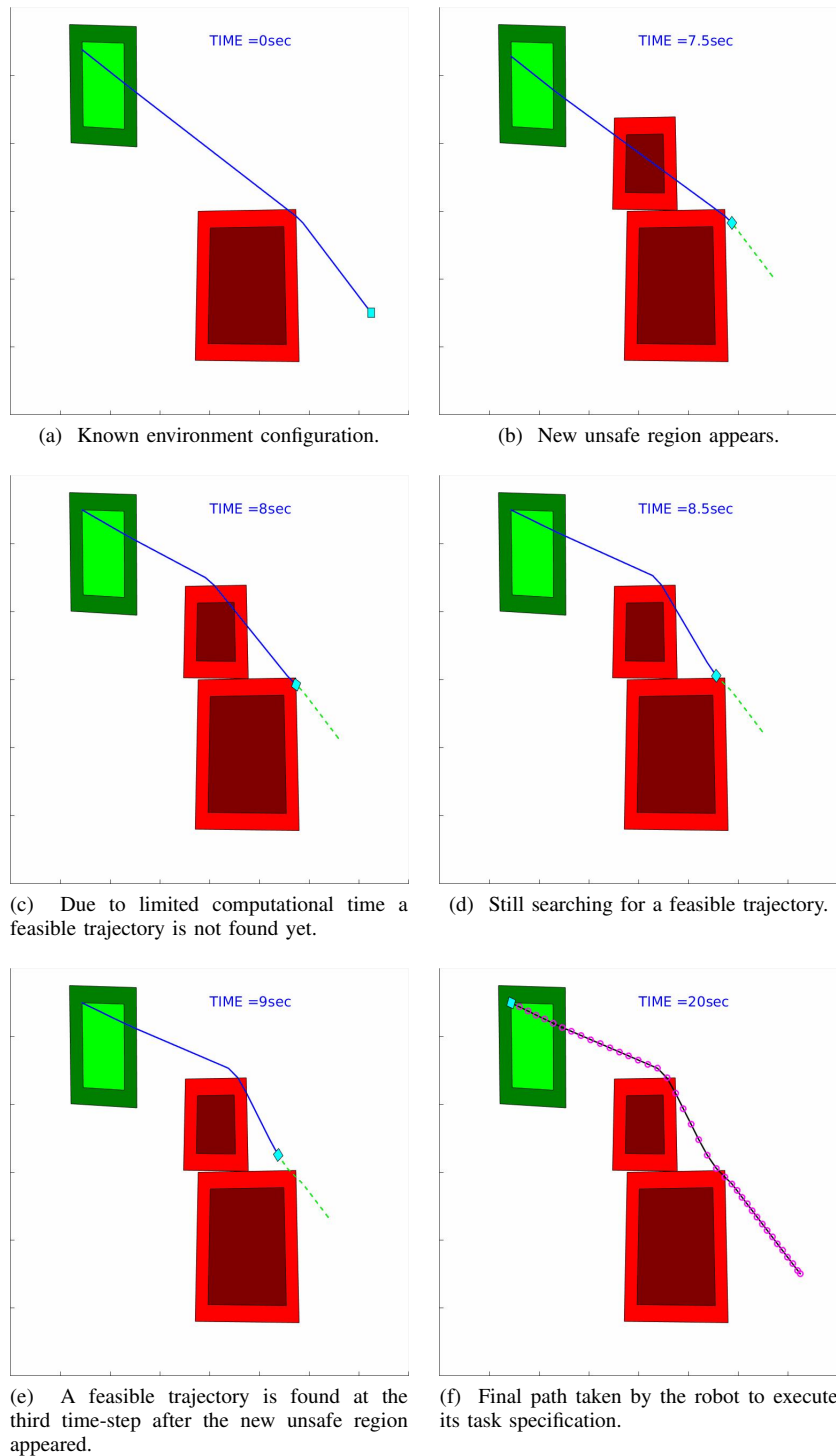
---

[4]http://tinyurl.com/sahaResearch#MILP-MTL

(a) Known environment configuration.

(b) New unsafe region appears.

(c) Due to limited computational time a feasible trajectory is not found yet.

(d) Still searching for a feasible trajectory.

(e) A feasible trajectory is found at the third time-step after the new unsafe region appeared.

(f) Final path taken by the robot to execute its task specification.

Figure 6. Real-time path planning in a dynamically changing environment.

VII. ACKNOWLEDGMENTS

REFERENCES

[1] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, 2000.

[2] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[3] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *Automatic Control, IEEE Transactions on*, vol. 53, no. 1, pp. 287–297, 2008.

[4] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*. IEEE, 2005, pp. 4885–4890.

[5] J. DeCastro and H. Kress-Gazit, "Synthesis of nonlinear continuous controllers for verifiably-correct high-level, reactive behaviors," *International Journal of Robotics Research Accepted*, 2014.

[6] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive, high-level robot control," *Robotics Automation Magazine, IEEE*, vol. 18, no. 3, pp. 65–74, Sept 2011.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications," in *American Control Conference (ACC), 2012*. IEEE, 2012, pp. 735–742.

[8] S. C. Livingston, E. M. Wolff, and R. M. Murray, "Cross-entropy temporal logic motion planning," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 269–278.

[9] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-uav mission planning," *International Journal of Robust and Nonlinear Control*, vol. 21, no. 12, pp. 1372–1395, 2011.

[10] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 5319–5325.

[11] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 2117–2122.

[12] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 3953–3958.

[13] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, S. Seshia *et al.*, "Model predictive control with signal temporal logic specifications," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 81–87.

[14] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *Automatic Control, IEEE Transactions on*, vol. 57, no. 11, pp. 2817–2830, 2012.

[15] A. Ulusoy, M. Marrazzo, and C. Belta, "Receding horizon control in dynamic environments from temporal logic specifications." in *Robotics: Science and Systems*, 2013.

[16] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 239–248.

[17] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[18] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[19] G. E. Fainekos and G. J. Pappas, *Robustness of temporal logic specifications*. Springer, 2006.

[20] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness. 1979," *San Francisco, LA: Freeman*, 1979.

[21] M. G. Earl and R. D'Andrea, "Iterative milp methods for vehicle-control problems," *Robotics, IEEE Transactions on*, vol. 21, no. 6, pp. 1158–1167, 2005.

[22] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[23] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Prentice Hall, 2002.

[24] A. A. Julius and A. K. Winn, "Safety controller synthesis using human generated trajectories: Nonlinear dynamics with feedback linearization and differential flatness," in *Proc. American Control Conference*, Montreal, Canada., 2012, pp. 709–714.

[25] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, *S-taliro: A tool for temporal logic falsification for hybrid systems*. Springer, 2011.

[26] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*. IEEE, 2004, pp. 284–289. [Online]. Available: http://users.isy.liu.se/johanl/yalmip