BluSTL: Controller Synthesis from Signal Temporal Logic Specifications

Alexandre Donzé[1] and Vasumathi Raman[2‡]

v0.1, 2015-03-14

Abstract

We present BluSTL, a MATLAB toolbox for automatically generating controllers from specifications written in Signal Temporal Logic (STL). The toolbox takes as input a system and a set of constraints expressed in STL and constructs an open-loop or a closed-loop (in a receding horizon or Model Predictive fashion) controller that enforces these constraints on the system while minimizing some cost function. The controller can also be made reactive or robust to some external input or disturbances. The toolbox is available at `https://github.com/vraman/BluSTL`.

1 Introduction

In [6, 5] we described a new technique for synthesizing controllers for hybrid systems subject to specifications expressed in Signal Temporal Logic (STL). The present document introduces the toolbox BluSTL, which implements the ideas presented in these papers. The toolbox takes as input a linear system (which can result from the linearization of some non-linear system), a set of constraints expressed in STL, and a cost function, and outputs a controller. The controller can be either in *open-loop*, i.e., it will compute a fixed sequence of inputs to be used by the system, or in *closed-loop* in a receding horizon fashion. In the latter case, a sequence of inputs is computed at each step, and only the first input values is used for one time step, and the process is reiterated. One specificity of the toolbox is that the user can tune the robustness of satisfaction of the STL specifications as defined in [1]. The toolbox also supports robust controller synthesis in more classical sense, i.e., robust to variations of some external disturbance input.

The approach as described in [6, 5] is based on encoding the system dynamics, the STL constraints and the cost function together in a Mixed-Integer Linear Problem (MILP). The controller then consists in a pre-compiled MILP which can be solved efficiently by modern MILP solvers, such as Gurobi [2]. Experiments show that while the pre-compilation phase, which can be done off-line, can take a significant time (several seconds to minutes, depending on the complexity of the dynamics and the specifications), the resulting problem can then generally be solved very quickly, which makes it possible to use the resulting controller on-line and possibly in real-time. The rest of the paper briefly describes the theoretical background and then presents a small tutorial example.

2 Some Theoretical Background

2.1 System dynamics

We consider a continuous-time system Σ of the form

*This work is supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

†[1]V. Raman is with the California Institute of Technology, Pasadena, CA, USA `vasu@caltech.edu`

‡A. Donzé is with the Department of Electrical Engineering and Computer Science, UC Berkeley, Berkeley, CA 94720, USA `donze@berkeley.edu`

$$\dot{x} = Ax + B_u u + B_w w \quad (1)$$
$$y = Cx + D_u u + D_w w \quad (2)$$

where

- $x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the *system state*,
- $u \in \mathcal{U} \subseteq \mathbb{R}^m$ is the *control input*,
- $w \in \mathcal{W} \subseteq \mathbb{R}^l$ is the *external input*,
- $y \in \mathcal{Y} \subseteq \mathbb{R}^o$ is the *system output*.

Given a sampling time $\Delta t > 0$, we discretize Σ into Σ_d of the form

$$x(t_{k+1}) = A^d x(t_k) + B_u^d u(t_k) + B_w^d w(t_k) \quad (3)$$
$$y(t_k) = C^d x(t_k) + D_u^d u(t_k) + D_w^d w(t_k) \quad (4)$$

where for all $k > 0$, $t_{k+1} - t_k = \Delta t$ and $t_0 = 0$. Given an integer $N > 0$, $x_0 \in \mathcal{X}$, and two sequences $\mathbf{u} \in \mathcal{U}^{N-1}$ and $\mathbf{w} \in \mathcal{W}^{N-1}$ noted

$$\mathbf{u} = u_0 u_1 \dots u_{N-1}$$
$$\mathbf{w} = w_0 w_1 \dots w_{N-1}$$

we denote by $\xi(x_0, \mathbf{u}, \mathbf{w}) \in \mathcal{X}^N$ the 4-uple of sequences $(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{w}) = \xi(x_0, \mathbf{u}, \mathbf{w})$ such that $\mathbf{x}$, $\mathbf{y}$, $\mathbf{u}$ and $\mathbf{w}$ satisfy (3-4) with $x(t_k) = x_k$, $y(t_k) = y_k$, $u(t_k) = u_k$ and $w(t_k) = w_k$ for all k. $\xi(x_0, \mathbf{u}, \mathbf{w})$, or sometimes simply ξ is called a run of Σ_d.

2.2 Signal Temporal Logic

We consider STL formulas defined recursively according to the grammar[1]

$$\varphi ::= \pi^\mu \mid \neg\psi \mid \varphi_1 \wedge \varphi_2 \mid \text{alw}_{[a,b]}\ \psi \mid \varphi_1\ \text{until}_{[a,b]}\ \varphi_2$$

where π^μ is an atomic predicate $\mathcal{X} \times \mathcal{Y} \times \mathcal{U} \times \mathcal{W} \to \mathbb{B}$ whose truth value is determined by the sign of a function $\mu : \mathcal{X} \times \mathcal{Y} \times \mathcal{U} \times \mathcal{W} \to \mathbb{R}$ and ψ is an STL formula. The fact that a run $\xi(x_0, \mathbf{u}, \mathbf{w})$ satisfies an STL formula φ is denoted by $\xi \models \varphi$. Informally, $\xi \models \text{alw}_{[a,b]}\varphi$ if φ holds at every time step between a and b, and $\xi \models \varphi\ \text{until}_{[a,b]}\ \psi$ if φ holds at every time step before ψ holds, and ψ holds at some time step between a and b. Additionally, we define $\text{ev}_{[a,b]}\varphi = \top\ \text{until}_{[a,b]}\ \varphi$, so that $\xi \models \text{ev}_{[a,b]}\varphi$ if φ holds at some time step between a and b. Formally, the validity of a formula φ with respect to the sequence $\mathbf{x}$ is defined inductively as follows

$$\begin{array}{lcl}
\xi \models \varphi & \Leftrightarrow & (\xi, t_0) \models \varphi \\
(\xi, t_k) \models \pi^\mu & \Leftrightarrow & \mu(x_k, y_k, u_k, w_k) > 0 \\
(\xi, t_k) \models \neg\psi & \Leftrightarrow & \neg((\xi, t_k) \models \psi) \\
(\xi, t_k) \models \varphi \wedge \psi & \Leftrightarrow & (\xi, t_k) \models \varphi \wedge (\xi, t_k) \models \psi \\
(\xi, t_k) \models \text{alw}_{[a,b]}\varphi & \Leftrightarrow & \forall t_{k'} \in [t_k + a, t_k + b], (\xi, t_{k'}) \models \varphi \\
(\xi, t_k) \models \varphi\ \text{until}_{[a,b]}\ \psi & \Leftrightarrow & \exists t_{k'} \in [t_k + a, t_k + b] \text{ s.t. } (\xi, t_{k'}) \models \psi \\
 & & \wedge \forall t_{k''} \in [t_k, t_{k'}], (\xi, t_{k''}) \models \varphi.
\end{array}$$

[1]For the readers familiar with temporal logic, note that the notation ev and alw (for $\Box$ and $\Diamond$) are taken from the syntax also implemented in the toolbox Breach.

An STL formula φ is *bounded-time* if it contains no unbounded operators; the *bound* of φ is the maximum over the sums of all nested upper bounds on the temporal operators, and provides a conservative maximum trajectory length required to decide its satisfiability. For example, for $\text{alw}_{[0,10]}\text{ev}_{[1,6]}\varphi$, a trajectory of length $N \geq 10 + 6 = 16$ is sufficient to determine whether the formula is satisfiable.

2.3 Robust Satisfaction of STL formulas

Quantitative or robust semantics defines a real-valued function ρ^φ of signal ξ and t such that $(\xi, t) \models \varphi \equiv \rho^\varphi(\xi, t) > 0$. In this work, it is defined as follows:

$$\begin{array}{lcl}
\rho^{\pi^\mu}(\xi, t_k) & = & \mu(x_k, y_k, u_k, w_k) \\
\rho^{\neg\psi}(\xi, t_k) & = & -\rho^{\psi}(\xi, t_k) \\
\rho^{\varphi_1 \wedge \varphi_2}(\xi, t_k) & = & \min(\rho^{\varphi_1}(\xi, t_k), \rho^{\varphi_2}(\xi, t_k)) \\
\rho^{\varphi_1 \vee \varphi_2}(\xi, t_k) & = & \max(\rho^{\varphi_1}(\xi, t_k), \rho^{\varphi_2}(\xi, t_k)) \\
\rho^{\text{alw}_{[a,b]}\psi}(\xi, t_k) & = & \min_{t_{k'} \in [t+a, t+b]} \rho^{\psi}(\xi, t_{k'}) \\
\rho^{\varphi_1 \text{ until}_{[a,b]} \varphi_2}(\xi, t_k) & = & \max_{t_{k'} \in [t+a, t+b]}(\min(\rho^{\varphi_2}(\xi, t_{k'}), \\
 & & \qquad \min_{t_{k''} \in [t_k, t_{k'}]} \rho^{\varphi_1}(\xi, t_{k''}))
\end{array}$$

To simplify notation, we denote ρ^{π^μ} by ρ^μ for the remainder of this document. The robustness of satisfaction for an arbitrary STL formula is computed recursively from the above semantics by propagating the values of the functions associated with each operand using min and max operators corresponding to the various STL operators. For example, the robust satisfaction of π^{μ_1} where $\mu_1(x) = x - 3 > 0$ at time 0 is $\rho^{\mu_1}(\xi, 0) = x_0 - 3$. The robust satisfaction of $\mu_1 \wedge \mu_2$ is the minimum of ρ^{μ_1} and ρ^{μ_2}. Temporal operators are treated as conjunctions and disjunctions along the time axis: since we deal with discrete time, the robustness of satisfaction of $\varphi = \text{alw}_{[0,2.1]}\mu_1$ is

$$\rho^\varphi(x, t) = \min_{t_k \in [0, 2.1]} \rho^{\mu_1}(x, t_k) = \min\{x_0 - 3, x_1 - 3, \ldots, x_K - 3\}$$

where $0 \leq t_0 < t_1 < \ldots < t_K \leq 2.1 < t_{K+1}$.

The robustness score $\rho^\varphi(\xi, t)$ should be interpreted as *how much* ξ satisfies φ. Its absolute value can be viewed as the distance of ξ from the set of trajectories satisfying or violating φ, in the space of projections with respect to the function μ that define the predicates of φ.

2.4 Controller Synthesis

Given an STL formula φ and a *cost function* of the form $J(x_0, \mathbf{u}, \mathbf{w}, \varphi) \in \mathbb{R}$, BluSTL can solve different control synthesis problem, either in *open loop* or *closed loop*, and with a *deterministic* or *adversarial* environment (robust control). In all problems, we assume given an initial state $x_0 \in \mathcal{X}$, an horizon L and some reference disturbance signal $\mathbf{w}^{\text{ref}} \in \mathcal{W}^N$.

Problem 1 (Open loop, deterministic) *Compute* $\mathbf{u}^* = u_0^* u_1^* \ldots u_{N-1}^*$ *where*

$$\begin{array}{rl}
\mathbf{u}^* = & \underset{\mathbf{u} \in \mathcal{U}^N}{\text{argmin}}\, J(x_0, \mathbf{u}, \mathbf{w}^{ref}, \varphi) \\
 & s.t.\ \xi(x_0, \mathbf{u}, \mathbf{w}_0) \models \varphi
\end{array}$$

Problem 2 (closed loop, deterministic) *Given an horizon* $0 < L < N$*, for all* $0 \leq k \leq N - L$*, compute* u_k^* *as the first element of the sequence* $\mathbf{u}_k^L* = u_k^L * u_{k+1}^L * \dots u_{k+L-1}^L*$ *satisfying*

$$\begin{aligned} \mathbf{u}_k^L* = \quad & \underset{\mathbf{u}_K^L \in \mathcal{U}^L}{\operatorname{argmin}} J(x_k, \mathbf{u}_k^L, \mathbf{w}_k^{ref}, \varphi) \\ & s.t.\ \xi(x_k, \mathbf{u}_k^L, \mathbf{w}_k^{ref}) \models \varphi \end{aligned}$$

These two problems both admit an adversarial version where $\mathbf{w}$ is allowed to vary in some region around $\mathbf{w}^{\text{ref}}$ while satisfying some constraints. In this case, BluSTL treats $\mathbf{w}$ as an adversary for the controller which tries to falsify φ. The control input returned by BluSTL, if any, is the first one after some iterations for which this falsification is infeasible. If no such input is found, BluSTL stops and declares the problem infeasible. We refer the reader to [5] for more details.

3 Getting Started

3.1 Installing BluSTL

BluSTL depends on YALMIP [4], which is best obtained with the Multi-Parametric toolbox (MPT3) [3]. Most experiments have been done with the Gurobi solver [2] as back-end, though other solvers might work as well. Once YALMIP (or MPT3) is installed, the only thing to do to install BluSTL is to add the path `BluSTL/src` to Matlab paths. In the following, we present a tutorial example for a simple double integrator system. The tutorial script can be found in `BluSTL/examples/tutorial1.m`.

3.2 A Small Tutorial

Defining the plant dynamics

The toolbox is organized around one main class, called STLClti. An STLClti object is primarily a continuous Linear Time Invariant (LTI) system with inputs, outputs and disturbances. Hence, a constructors for this class takes matrices to define such an LTI. We first define and A and B matrices for state evolution:

```
A = [0 1 ;
     0 0];
Bu = [0;1];
```

Later on, we will use a disturbance signals so we need to define a Bw matrice. This signal will not influence the state dynamics, though, so we set Bw to be 0.

```
Bw = [0;0];
```

Next we define the output dynamics, i.e., C, Du and Dw matrices. Here we have a single output $y(t) = x_1(t)$.

```
C = [1 0];
Du = 0;
Dw = 0;
```

Now we can call the main constructor of STLC_lti class.

```
Sys= STLC_lti(A,Bu,Bw,C,Du,Dw);
```

In the next section, we will define the different settings for the control synthesis experiment. Before that, we define some initial state:

```
Sys.x0= [1 ; 1];
```

Defining the controller

We start by defining the time instants for the whole experiment, the discrete time step ts for the controller and the horizon L in number of time steps.

```
Sys.time = 0:.1:10;
Sys.ts=.2; % sampling time for controller
Sys.L=10;  % horizon is 2s in that case
```

Next we declare some constraints on control inputs, here, lower and upper bounds:

```
Sys.u_ub = 10;  % upper bound on u
Sys.u_lb = -10; % lower bound on u
```

Then the following define a signal temporal logic formula to be satisfied by the system. Note that times in the temporal operators are continuous, not discrete steps.The following formula specifies that the output signal must remain at distance 0.1 from the external signal for 0.5 seconds after at most 1 second.

```
Sys.stl_list = {'ev_[0,1.] alw_[0,0.5] ( abs(y1(t)-w1(t)) < 0.1)'};
```

Now we are ready to compile the controller for our problem.

```
controller = get_controller(Sys)
```

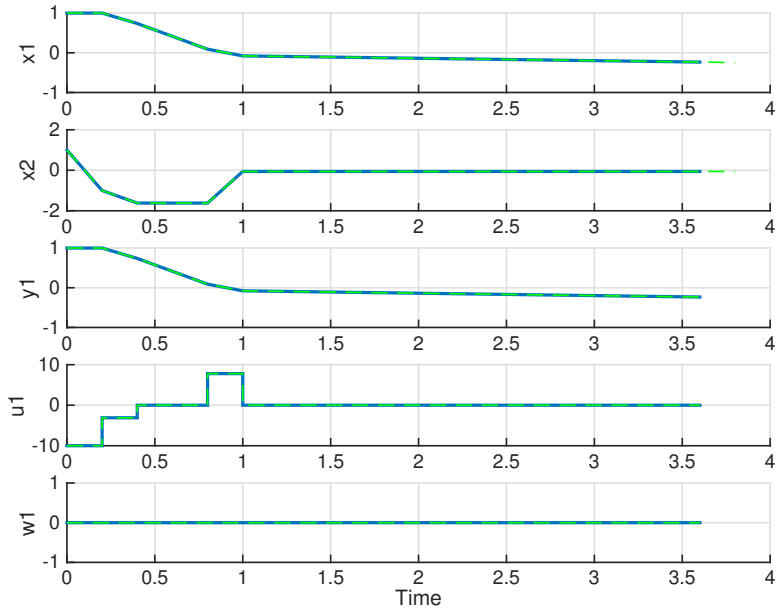
```
Optimizer object with 108 inputs and 79 outputs. Solver: GUROBI-GUROBI
```

Note that by default, the objective function will minimize the 1-norm of the input.

Testing the controller

The simplest mode to run our system with the newly created controller is in open loop. This is done with the following command:

```
run_open_loop(Sys, controller);
```



We can run our system in closed loop, but this is not very interesting, because w is 0 anyway. Let's change it to take value 1 between time 3 and 4 and -0.5 between time 6 and 8:

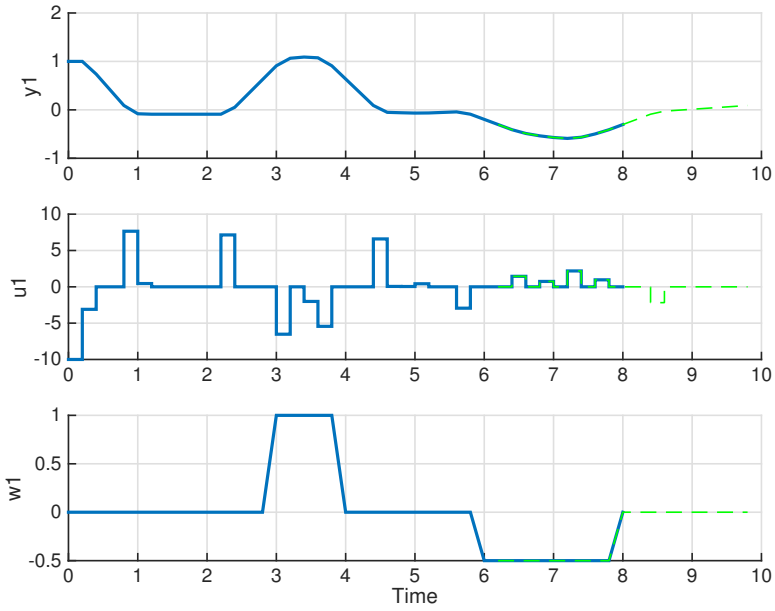
```
Sys.Wref = Sys.time*0.;
Sys.Wref(30:40) = 1;
Sys.Wref(60:80) = -0.5;
```

Now we change the specification to an unbounded horizon one, where at all instant, the output must track the external signal with some specified maximum delay:

```
Sys.stl_list = {'alw (ev_[0,1.] alw_[0,0.5] ( abs(y1(t)-w1(t)) < 0.1))'};
controller = get_controller(Sys);
```

This time we will only plot input and outputs, i.e., disable the state plotting:

```
Sys.plot_x =[]; % default was Sys.plot_x = [1 2]
run_deterministic(Sys, controller);
```



More examples are given in the folder BluSTL/examples. In particular the hvac_room case study demonstrate the adversarial scenario, as well as plot customization. The idea is to create a class derived from STLC_lti and specialize the update_plot method.

4 Future Work

BluSTL v0.1 is still at an early stage of development, and beside the usual bug fixes, performance and stability issues, it can be further improved in many directions. One limitation is that unbounded specifications are currently limited to alw(φ) where φ is a bounded horizon formula, where the horizon should be smaller than $L \times ts$. We are working on supporting more unbounded horizon specifications, including ev and until with possible nesting (e.g. ev alw or alw ev. Another direction is to lift the constraint on the horizon of sub-formulas, by using other semantics, e.g., *weak* semantics, adapted to partial traces. Another limitation is on the type of systems considered. It is relatively easy to use BluSTL for any kind of systems that admit a proper linearization, but the linear models used in BluSTL are fixed. It would be interesting to implement controller with switched linear dynamics, or more general hybrid models such as, e.g., Mixed Logical Dynamics used in the MPT.

References

[1] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, pages 92–106, 2010.

[2] I. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

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

[4] J. Lfberg. Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[5] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the international conference on Hybrid Systems: Computation and Control, HSCC 2015*, 2015.

[6] V. Raman, M. Maasoumy, A. Donzé, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia. Model predictive control with signal temporal logic specifications. In *Proc. of the IEEE Conf. on Decision and Control*, 2014.