Verification of Hierarchical Control via Approximate Simulation and Feedback Linearization

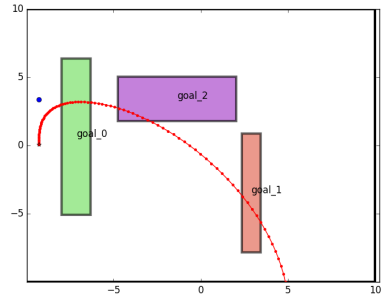
Andres Cabrera-Flor, Joseph McMahon, and Nishan Srishankar

***Abstract*—The project is to verify approximate simulations of hierarchical controllers for high-order nonlinear systems that are feedback linearizable. Recent work has been done by J. Fu, [1] (2013) and G. J. Pappas [2] (2000) on the production of low-order linear controllers via approximate simulation relation, which is the basis for the project. Creating low-order specifications that guarantee correctness under unknown disturbances in the original high-order system remains an open scientific question. Progress will be made towards verification by considering the simplified scaling chains of integrators problem, and iteratively adding complexity to it. For example, obstacles were ignored.**

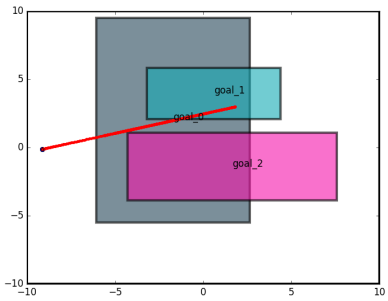
I. Progress Made

The motivation of the project is to make progress towards the open question of verifying hierarchical controllers, it is important to document the steps planned and completed, and describe what remains for future work.

1) All of the open-source Python libraries are installed and working on all three team members' Ubuntu installations. Team WPI's submission to ICRA 2016's Fmrchallenge also works.
2) The Π-related low-order system was automatically generated from the original high-order system using our HCA toolbox. [2] The Π-related systems from the Fu paper and Pappas papers are described in the Dynamics and Controls sections. After getting the low-order gain matrix K from the LQR controller, there was no way to simulate it back in the high-order system due to the ROS architecture of Fmrbenchmark. This is to enforce that your controller behaves as specified. To program the Π-related simulation, it was attempted that another Instance Monitor was spun up with the same Problem Instance (i.e. $n = 2$, $m = 3$), but a lower number of integrators (i.e. $m = 1$). Finally, it was realized too late that an additional ROS topic was the better solution.
3) A key observation made while implementing the HCA toolbox was that by changing the gain matrix K output by the LQR controller to favor velocity and acceleration over position, much faster simulation times could be achieved as shown in 1.
4) Tulip was attempted to be used to automate the generation of the high and low-order systems' LTL specifications and Omega automata. There were again ROS software engineering issues that made it difficult to integrate Fmrbenchmark with Tulip, and Polytope has a known bug as well. [13]



(a) First LQR with HCA-influenced gain matrix K.



(b) Second LQR with HCA-influenced gain matrix K.

Fig. 1: Using the Π-related system created from the HCA toolbox, some interesting and speedy results were obtained.

II. Introduction

Optimal control is easily achieved for low-order linear systems with controllers such as PID or LQR, whereas high-order systems are more accurate models. However, generating controllers from high-order specifications is computationally complex. [2]

If low-order linear specifications and their automatically generated controllers can guarantee correctness of the original high-order specification, it is very valuable. [1] This is because most systems can be reformulated into high-order systems, and then verified low-order controllers automatically generated. [8]

The high-order specifications shown below in equations 1 and 2 are typical Linear Temporal Logic constraints. A point-robot must visit all of the goal regions and avoid all of the obstacle regions.

$$q(0) = init \wedge \Box(q(t) \notin Obs) \wedge \bigwedge_i \Box\Diamond goal_i \qquad (1)$$

In constraint equation (2), a time constraint is added.

$$q(0) = init \wedge \Box(q(t) \notin Obs) \wedge \bigwedge_i (counter_i \not\leqslant T_i) \bigcup goal_i \tag{2}$$

where $Obs \subset \mathbb{R}^n$; $init \in \mathbb{R}^n$.

Sample 2D and 3D trajectories are shown below in Figures 2, 3, and 4.

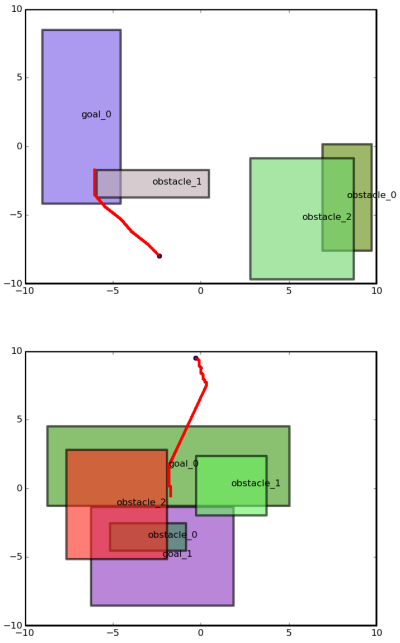


Fig. 2: LQR tracking A* generated paths.

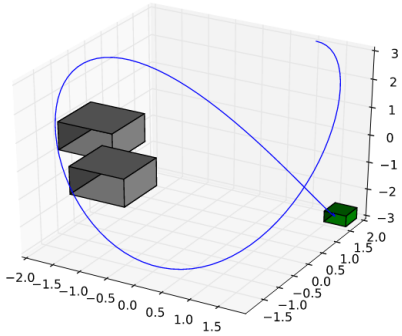


Fig. 3: Visualization of a path in 3D state space with obstacles.

This is a good team to work on this approach to the verification problem because of Joseph McMahon and Nishan Srishankar's previous work with chains of integrators for the ICRA 2016 Fmrchallenge in Stockholm, Sweden. [11] Previously written code has already been integrated with Scott Livingston's open-source frameworks (Fmrbenchmark, Tulip, Polytope) and may be leveraged to speed up testing. [12] The team also wants to contribute to these open-source Python libraries in the near future if possible.

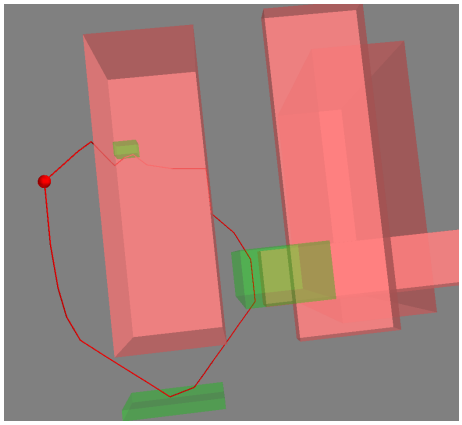


Fig. 4: A* and LQR controlling a point robot in 3D state space.

A. *Problem Statement*

The goal is for the point robot to move through the goal regions without hitting obstacles. This requires trajectory generation, motion planning, and controls. However, to make the verification problem more approachable, obstacles are being ignored at first, and will be reimplemented as the last step of the approach.

The temporal logic equation can be depicted as an automaton for $i = 1$ as shown in Figure 5.

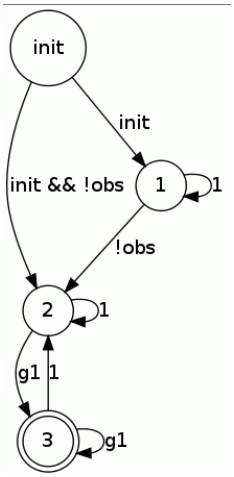


Fig. 5: Buchi Automaton for given LTL equation for one goal and one obstacle.

Two controllers are bisimilar when they both have the same transition system (identical labels). Furthermore, for one goal and one obstacle, the controllers both satisfy the specifications if they conform to the Buchi Automaton in Figure 5.

The inputs of the project are: high-order systems and specifications, and several disturbance thresholds.

The outputs of the project are: minimum-order linear systems, specifications, and controllers that guarantee correctness in the original high-order system.

III. Methodology

The complexity of the problem requires it to be simplified, and then extended later. This will be done by considering linear systems instead of nonlinear feedback linearizable systems. The methodology is broken into three main steps, which serve as the expected project goals.

A. Expected Goals

The expected goals are what the team believes it can accomplish during the Fall 2016 semester. Four of the five expected goals were met by following the Approach section.

1) Generate Π-related low-order systems from high-order chain of integrators.
2) Control the low-order system in the high-order state space while ignoring obstacles.
3) Compare output of low-order LQR trajectories with original high-order state space and specification.
4) Automate the generation of LTL specifications and Omega automata.
5) Switch LQR controller to Tulip's Control2Facet controller and motion planner. This is the only expected goal that was not met.

B. Reach Goals

It is not expected that the team will complete any of the reach goals during the Fall 2016 semester, but they are important to show the next steps and general logic of the approach. They may be implemented in the future as part of directed research or a doctorate study.

1) Add disturbances to the high-order system and control it. Compare to low-order system results.
2) Add obstacles to the high-order system and control it. Compare to low-order system results.
3) Update low-order specifications to satisfy high-order specifications under the disturbance threshold.
4) Extend the work to nonlinear feedback linearizable systems (or intermediate class of systems).

If time allows, the project work will be extended to nonlinear systems. It may not be practical to extend the linear work directly to nonlinear feedback linearizable systems, and there may be an easier subset of nonlinear systems to work with. For example, nonlinear systems that "enjoy a structural property known as differential flatness" [1] are easy to work with because they are made up ordinary differential equations which are easier to linearize.

IV. Approach

The first thing done was to refamiliarize the team with Fmrbenchmark and Team WPI's ICRA 2016 Fmrchallenge code, and setup a new coding environment for the team to work on. There were a few installation problems with Polytope early on. After this, obstacles were removed by creating a new configuration file in the integrator chains dynamaestro.

The HCA function [2] was made into a new toolbox, the HCA toolbox, which calculates the new low order system dynamics F and G from the high order system dynamics A and B. Π-relation system generated and ran simultaneously to the original high-order system simulation. The low and high-order results were overlaid and compared. The LTL specification and Omega automata were generated for both the low and high-order systems.

The next steps are to finish implementing Tulip's Control2Facet controller and motion planner, and then to add noise and disturbances in, and finally add obstacles back in.

It will be difficult to characterize the behavior around obstacles due to disturbances, and that is the main challenge of the entire project. However, a large amount of work must be completed prior to this.

1) Generate Π-related low-order system from high-order chain of integrators.
2) Control the low-order system in the high-order state space using the Π-relation from step 1 and LQR control. Ignore obstacles.
3) Compare output and specification of low-order LQR trajectories from step 2 with high-order output and specification.
4) Use Control2Facet to control the low-order system.
    a) Generate LTL.
    b) Synthesize controller.
    c) Generate Omega Automata.
5) Add disturbance to the high-order system. Update low-order specifications to satisfy high-order specifications under the new disturbance threshold.
6) Add obstacles back in.

Steps 1-4 were worked on during the Fall 2016 semester. One major issue was the amount of time it took to work with the Python libraries, since there is relatively little documentation and almost no active community. These issues only serve to justify why these open-source Python libraries deserve more attention and contribution.

A. Open-source Python Libraries

The toolkits and programming languages used can affect the results and execution of a analytical project such as this. It is necessary to explain why the selected libraries and Python were chosen.

The Python language and the various open-source libraries for it lend itself well to benchmarking problems, because of the freely available source code. The industry can agree on a set of benchmarking tools, and easily compare results. The competing alternative is Matlab, which is a well-established proprietary software system. Although Matlab has robust toolkits and active community, it is not suitable or adaptable enough to serve an industry-wide robotics benchmarking application.

The primary libraries were Fmrbenchmark and Tulip. They are described below, as well as their core dependencies.

- **Rospy** (http://wiki.ros.org/rospy) is a Python wrapper for Roscpp, which is C++ Robot Operating System ROS (http://wiki.ros.org/) package manager. ROS is an industry standardized common adapter for open-source and proprietary robotics libraries.

- **Fmrbenchmark** is a robotics benchmarking tool written by Scott Livingston and Vasumathi Ramen under Richard Murray at Caltech. (https://fmrchallenge.org/, https://github.com/fmrchallenge/fmrbenchmark) It has three common domains: Scaling chains of integrators, Traffic network of Dubins cars, and Factory cart clearing. It is a good benchmarking framework because of its ability to accurately handle errors, noises, and disturbances in near-continuous systems.
- **Python-control** is a controller toolbox maintained by Scott Livingston (https://github.com/python-control/python-control) that is only being used for the Linear Quadratic Regulator that comes default in Fmrbenchmark. One goal is to switch out the LQR controller for Tulip's Control2Facet controller. The LQR controller is omitted from this report because it is well-known and well-documented.
- **Tulip-control** is a large FMR toolbox primarily written by Scott Livingston. It contains the Control2Facet functionality without explicitly stating that it does (https://github.com/tulip-control/tulip-control, http://tulip-control.sourceforge.net/, http://tulip-control.sourceforge.net/doc/install.html).
- **Gr1c** is GR(1) synthesis for LTL specifications which is primarily used by Tulip (https://github.com/tulip-control/gr1c).
- **Polytope** is a tool for easily representing goals and obstacles as polytopes. This implementation uses very little memory to represent polytopes, which helps functions that take polytopes as inputs be very fast.
- **Team WPI's submission to ICRA 2016 Fmrchallenge** contains 3D plotting, TSP, other helper functions that are already integrated with Fmrbenchmark.

V. Dynamics

Because high-order linear systems are the first step, an analysis of linear systems is presented.

From the primary paper [1], the following Π-related method is used to find an abstraction of a given higher-order linear system:

$$\Sigma_1 : \dot{x} = Ax + Bu$$

$$y = Cx$$

Where the state matrix is $\mathbb{R}^n$, the input and output matrices are $\mathbb{R}^m$. Additionally, it is assumed that the given higher-order system is stable i.e. eigenvalues of (A-B*K) will have negative real parts. A second, lower-order system is presented to be:

$$\Sigma_2 : \dot{z} = Fz + Gv$$

$$w = Hz$$

In this case, the abstraction has a state matrix $\mathbb{R}^p$, an input matrix $\mathbb{R}^k$ and an output $\mathbb{R}^k$. By definition, $p \leq n$ as the abstraction will have a lower rank that the given system. Σ_2 is Π-related to Σ_1 if the following hold:

$$\Pi[Ax + Bu] = F\Pi x + Gv$$

$$C = H\pi$$

$$w(t) = \Pi x(t)$$

where Π is a $p \times n$ matrix and $ker(\Pi) = Im(\Pi)$

The approximated system matrices then become:

$$F = \Pi \times A \times \Pi^+$$

$$G = \Pi \times A \times B$$

Π^+ is the pseudo-inverse of Π obtained by Singular-Value-Decomposition, and equals $\Pi^T(\Pi \times \Pi^T)^{-1}$.

Our initial idea is to approximate a higher order chain of integrators with that of a single integrator using the method shown above. The dynamics of the system are relatively simple as the problem requires controlling a point robot. A two dimensional system (single integrator) can be expressed as a linear system of equations given below:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \xi$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} + \eta$$

where ξ and η are either bounded disturbances (non-deterministic) or stochastic processes.

The system given by the differential equation $D^m q = u$ is called a chain of integrators because it can be written as a series of linear control equations as shown below.

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= x_3 \\ &... \\ \dot{x}_{m-1} &= x_m \\ \dot{x}_m &= u \\ y &= x_1 \end{aligned}$$

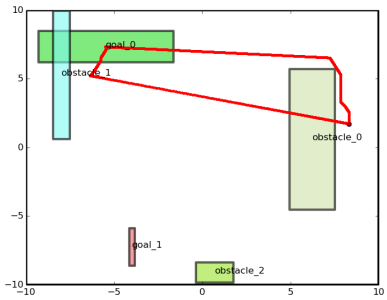
VI. Controls

The Π-relation will be used to generate a low-order system which can be easily controlled along a trajectory. The Control2Facet methodology will then be used to do motion planning and control the low-order system.

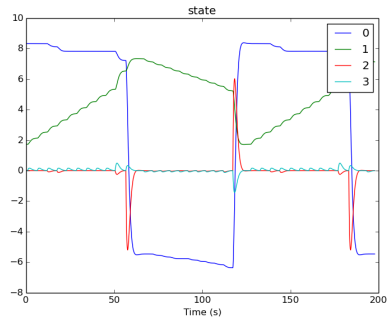
Although the LQR controller is being used in the early steps, it is not reviewed because there is an abundance of prior art and information on the LQR controller.

A. Π-relation Controller

The Π-related system is trivial to generate because both the high and low-order systems are linear scales of integrator chains. The big issue is programming the simulators to run concurrently, or even consecutively.



(a) Trajectory of LQR tracking A* path.



(b) Evolution of State Vector while LQR tracks A* path.

Fig. 6: Simulation of LQR tracking A* generated path. It successfully avoids one obstacle and reaches one goal region.

VII. Motion Planning

Motion planning will also be achieved using the Control2Facet methodology. [7] This method first breaks the state space into simplexes (triangles in 2D, tetrahedra in 3D), which are formed with the corners of the goal and obstacle regions, and are within the bounds of the state space.

The simplexes that must be visited to satisfy the specification are identified. The facets between these simplexes each require a controller that guarantees any initial condition with the first simplex will result in the point robot exiting the required facet.

VIII. Results

Progress was made on the first four item from the Approach section, the completion of those tasks and steps 5-6 remain for Future Work.

1) Generate Π-related low-order system from high-order chain of integrators. Progress was made in automatically generating the Π-related low-order system dynamics F and G from the high-order system dynamics A and B. This was done by creating a new HCA toolbox from the pseudo-code in the Pappas paper [2]. The issue then, however, is that the new low=order LQR controller outputs a gain matrix K that cannot be used on the original high-order state space $u = -Kx$. Although the Fmrbenchmark default LQR code uses $-K$, it was observed that the gain matrix scale from the HCA toolbox needed to be positive so that $u = Kx$.
2) Control the low-order system in the high-order state space using the Π-relation from step 1 and LQR control. Ignore obstacles. There was some surprise success in editing the gain matrix K with the output from the HCA toolbox as shown on Page 1 in 1. However, in general the two plots could not be overlaid on top of each other due to software engineering issues. Ignoring obstacles was no problem and was as simple as making a new configuration file and running $catkin_makeinstall$.
3) Compare output and specification of low-order LQR trajectories from step 2. with high-order output and specification. Without obstacles, error or disturbances, there is no reason that either system should be difficult to control.
4) Use Control2Facet to control the low-order system. Two attempts were made at implementing Tulip to automatically generate the formatted LTL specifications and synthesize the Omega automata. The first was based on the $continuous.py$ Tulip example, and the second was based on the $continuous_with_simulation.py$ example. The $continuous.py$ example was inserted into the main() function of Fmrbenchmark's default LQR controller The second attempt was the reverse, inserting Fmrbenchmark code into Tulip code. The abstraction input was generated but again could not be easily integrated with FMRBenchmark. The result was that we were informed that a bug in Polytope needs to be addressed. [13]
    a) Generate LTL. The LTL Specification format for Tulip is defined in the documentation. This specification is then synthesized using GR(1) LTL synthesis.
    b) Synthesize controller.
    c) Generate Omega Automata.

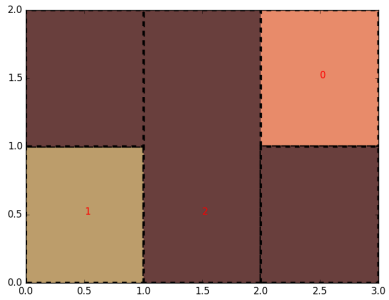
Using the scale of integrators problem and framework offered an approachable and testable method to hierarchical controller verification. This project helped refine the steps that need to be taken towards the final goal of verifying reduced nonlinear systems.

IX. Conclusion

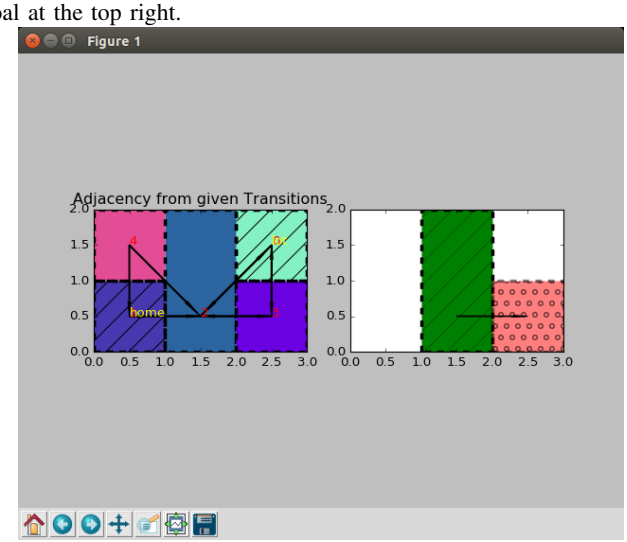
This project required Formal Methods of Robotics, Linear Algebra, and Software Engineering skills which were respectively filled by N. Srishankar, A. Cabrera, and J. McMahon.

In the **Formal Methods of Robotics** domain, more time should have been devoted to exploring Tulip's features early on, but installation problems did not get fixed and Fmrbenchmark was the primary focus. Fmrbenchmark itself is an impressive system, but there were a large number of problems integrating it with Tulip. For example, after passing variables like the system dynamics from Fmrbenchmark to Tulip, the ROS architecture of Fmrbenchmark completely supresses Tulip. The proper way to engineer it would have been to create a new ROS topic using Tulip, and have it listen to Fmrbenchmark as the trials are generated. This would also allow it to be run on a spread thread.

Furthermore, the concept of threading in ROS is an important one, because it is the same problem we had in



(a) Tulip continuous state space divided into home, free space, and one goal at the top right.



(b) Screenshot of Tulip performing facet transition and controller synthesis.

Fig. 7: Tulip synthesizes controllers for continuous state space linear dynamical systems.

the **Linear Algebra** domain after implementing the HCA toolbox for Approach step 1. If ROS topics can be managed properly, multiple scaling chains of integrators problems can be simulated concurrently (original high-order and low-order approximation at the same time). This capability would have profound impact on our ability to produce better results.

Finally, Software Engineering time was not accounted for sufficiently. More time should have been spent on the approaches mentioned earlier, but then again many of those problems were unknown unknowns of the beginning of the Fall 2016 semester.

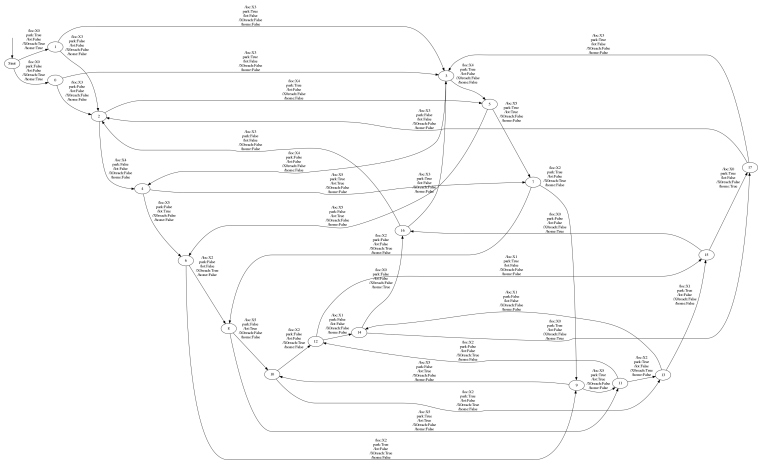
A. *Future Work*

Future Work remains in completing steps 1-4 and contributing to any open-source Python libraries required to make it happen. After that, the approach should be continued with steps 5-6 below.

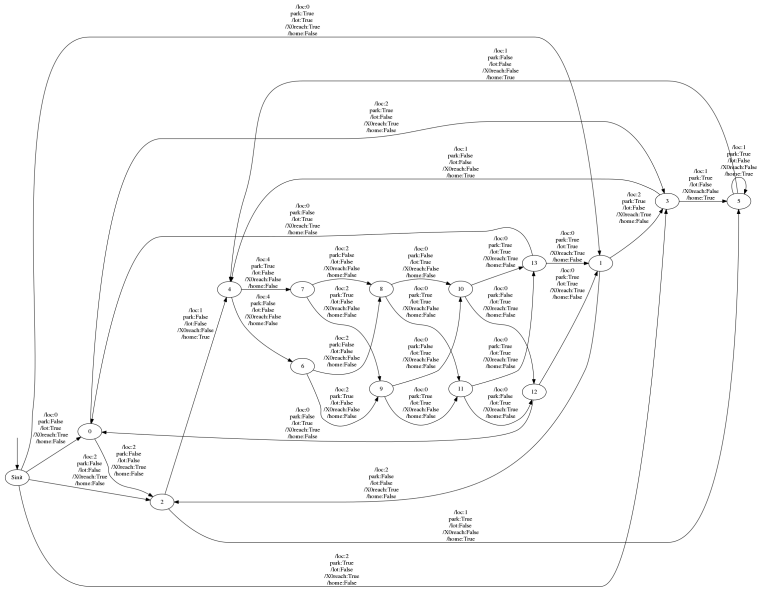
1) Add disturbance to the high-order system. Update low-order specifications to satisfy high-order specifications under the new disturbance threshold.
2) Add obstacles back in.

REFERENCES

[1] Fu, J., S. Shah, and H. G. Tanner, “Hierarchical control via approximate simulation and feedback linearization, ” Amer. Control Conf., Washington, DC, pp. 1816–1821, 2013.
[2] G. J. Pappas, G. Lafferriere, and S. Sastry, ”Hierarchically consistent control systems, ” IEEE Transactions on Automatic Control, vol. 45, pp. 1144-1160, 2000.
[3] P. Tabuada and G. J. Pappas, ”Hierarchical trajectory generation for a class of nonlinear systems, ” Automatica, vol. 41, pp. 701-708, 2005.
[4] A. Girard and G. J. Pappas, ”Hierarchical control system design using approximate simulation, ” Automatica, vol. 45, no. 2, pp. 566-571, 2009.
[5] G. Pappas and S. Simic, ”Consistent abstractions of affine control systems, ” IEEE Transactions on Automatic Control, vol. 47, no. 5, pp. 745-756, May 2002.
[6] A. Girard and G. J. Pappas, ”Approximation metrics for discrete and continuous systems, ” IEEE Transactions on Automatic Control, vol. 52, no. 5, pp. 782-798, May 2007.
[7] L.C.G.J.M. Habets, J.H. van Schuppen, ”A control problem for affine dynamical systems on a full-dimensional polytope, ” Automatica, vol. 40, no. 1, pp. 21-35, January 2004.
[8] M. Kloetzer, and C. Belta. ”A fully automated framework for control of linear systems from temporal logic specifications, ” IEEE Transactions on Automatic Control, vol. 53, no .1, pp. 287-297, 2008.
[9] Y. Kaihong and J. Haibo, ”Hierarchical control of linear systems from the abstraction feedback gain, ” Asian Journal of Control, vol. 18, no. 6, pp. 1–8, November 2016.
[10] R. Murray, ”Optimization-based control, ” 2nd ed. California Institute of Technology, 2010.
[11] R. Sheth, N. Srishankar, J. McMahon, and S. Srivastava, ”RBE 502 - Scaling Chains of Integrators, ” Worcester Polytechnic Institute, Department of Robotics Engineering, RBE 502, Robot Controls, Spring 2016, Dr. Jie Fu, April 2016.
[12] S. Livingston, ”The Temporal Logic Planning (TuLiP) toolbox, ” California Institute of Technology, The Regents of the University of Michigan, 2010-2016. http://tulip-control.sourceforge.net/
[13] Issue #170: *tulip-control/tulip-control*. (2016). GitHub. Retrieved 15 December 2016, from https://github.com/tulip-control/tulip-control/issues/170#issuecomment-267192059



(a) Discrete state space GR(1) Omega automaton.



(b) Continuous state space GR(1) Omega automaton.

Fig. 8: Tulip outputs GR(1) Omega automata.