Automated Fault Tolerant Control Synthesis based on Discrete Games

Grunnet, Jacob Deleuran; Bendtsen, Jan Dimon; Bak, Thomas

Published in:
I E E E Conference on Decision and Control. Proceedings

DOI (link to publication from Publisher):
10.1109/CDC.2009.5400374

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Automated Fault Tolerant Control Synthesis based on Discrete Games

Jacob D. Grunnet, Jan D. Bendtsen and Thomas Bak

***Abstract*— This paper focuses on how fault tolerant controllers can be designed based on discrete game abstractions of piecewise-affine hybrid systems (PAHS). The proposed method aims at automatic generation of the controllers by converting the discrete games to timed games, which can be solved by the UppAal toolbox. Winning strategies to the games are then refined to piecewise-affine control strategies which is a type of gain scheduling. The feasibility of the method is shown through the automated design of a fault tolerant controller for a simple example.**

I. Introduction

Controller design is basically an exercise in limiting a systems behaviour to a desired subset of the possible behaviours as specified by the requirements. An additional challenges lies in ensuring that the specifications are fulfilled while the system is influenced by e.g. unmodelled dynamics or unpredictable events such as component or subsystem failures.

The effect of components malfunction on closed loop systems is often highly disruptive and can adversely affect the systems performance. The fact that it is very hard to completely eliminate such failures has led to extensive research in Fault Tolerant Control (FTC) systems, see e.g. [1], [2]. Common for several of these strategies are that they view the problem as a game, where the designed controller plays against a malicious opponent controlling when and which faults occurs. For instance [3] relies on a robust control design methods which derive from the theory of dynamic games [4]. The result is a passive design where the controller guarantees stability and performance after the occurrence of a fault event. Other approaches e.g. [5] uses reconfiguration of controllers based on real-time detection of faults commonly referred to as Fault Detection and Isolation (FDI) [6].

The main task when designing strategies relying on reconfiguration is synthesising controllers for all possible failure modes. This synthesis can be carried out on-line for active FTC strategies or for passive control strategies off-line where a catalogue of controllers are computed a priori, see [7], [8]. On-line computation requires a fully automated synthesis method and adequate computing power, whereas off-line methods can only handle failure modes known at design time and can lead to large controller catalogues if a high number of faults and their combinations are to be handled.

This paper presents a passive FTC approach relying on solutions to discrete games representing the system in question. The method is based on recent advances in controller synthesis for piecewise-affine systems (see e.g. [9]–[13]). Particularly the work builds upon computing discrete abstractions of Piecewise-Affine Hybrid System (PAHS) [14], which is based on control to facet properties of affine systems on simplices as established in [9], [15]. The ultimate goal is to develop automated control synthesis procedures and the method proposed can be divided into 4 steps: Modelling the system including faults as a PAHS, computing a discrete game abstraction, finding a winning strategy to the discrete game and refining the resulting set of rules to control laws on the original system. In this paper the focus is on how to find winning strategies to discrete abstractions by mapping the problem to existing tools. A demonstration of the method using a simple example using double integrator dynamics and a single actuator fault is presented, showing the fully automated control synthesis based on a PAHS model of the system.

J. Grunnet, J. Bendtsen and T. Bak are with the Department of Electronic Systems, Section for Automation and Control, Aalborg University, Denmark. {grunnet,dimon,tb}@es.aau.dk

II. Methodology

In this work we consider discrete abstractions of PAHS representing systems including failure modes. It is assumed that discrete abstractions can be computed as shown in [16], [17], but for the sake of completeness a very short introduction to computing discrete game abstractions is given here.

A. Computing Discrete Abstractions

As shown in Fig. 1a the first step is to construct a model of the system and its, O, failure/operational modes, $M = \{S_1, S_2, \ldots, S_O\}$, defined as a set of piecewise-affine systems (PAS), $S \in M$,one for each mode. Each piecewise-affine system, $S = (L', K, F)$ denotes a finite affine linear simplicial switched system with control, where $L' \subset L = \mathbb{R}^n$ is a bounded polyhedral set (a polytope) of dimension n, $K = \{\Delta_i\}_{i \in I}$ is a simplicial partition of L' with a finite index set I and $F = \{f_i\}_{i \in I}$ is a family of affine linear functions

$$f_i : V_i \times \mathcal{U}_i \rightarrow L, \quad f_i(x, u) = \dot{x} = A_i x + B_i u + C_i \quad ,$$

with V_i an open neighbourhood of Δ_i and with the input set $\mathcal{U}_i \subseteq \mathbb{R}^m$ a polytope.

The PASs are connected via the map $T : M \times E \rightarrow M$, where E is a set of external events that can be thought of as the occurrence faults if they are uncontrollable and as operational modes if they are controllable. The resulting system is a PAHS with the possibility of uncontrollable events as defined in [18], see Fig. 1b. It should be noted that both guards and reset maps can be associated with the transitions but for many systems this is not necessary e.g.

for faults the guards are typically always enabled and the reset maps are the identity map as the state spaces, L', are identical for both PASs and no sudden change in the continuous state happens when a fault occurs. Notice that a fault does not necessarily alter the system dynamics, but could alter the bounded input space $\mathcal{U}$ signifying a loss of actuator performance.

By computing discrete abstractions for each PAS in the PAHS and connecting the abstractions according to the mode transitions a discrete abstraction of a PAHS can be obtained. The discrete abstractions of PAS are found by considering each simplex of a PAS as a discrete location and computing the transitions between the locations. Fig. 1d and 1c illustrates the result when the reset maps are identity maps and the discrete equivalents of simplices in the *Nominal* mode are therefore connected to the same simplex in the *Fault* mode.

When computing the discrete equivalent of a simplex, Δ_i, each facet is a potential transition with one of three properties:

Blocked	A control law exists which can guarantee that the system will not leave Δ_i through the facet.
Uncontrollable	The facet can not be blocked.
Controllable	A control law exists which guarantees that the facet can be blocked and conversely that it is possible to leave the Δ_i in finite time possibly through that facet.

The conditions associated with blockability and controllability can be described as LMIs [9]. Informally blockability of a facet can be described as the possibility to find inputs $u_j \in \mathcal{U}$ at all the $j = 1 \dots n$ vertices of a facet $p \in \Delta_i$ such that the derivatives, $\dot{x}_j$, points into the hull of Δ_i.

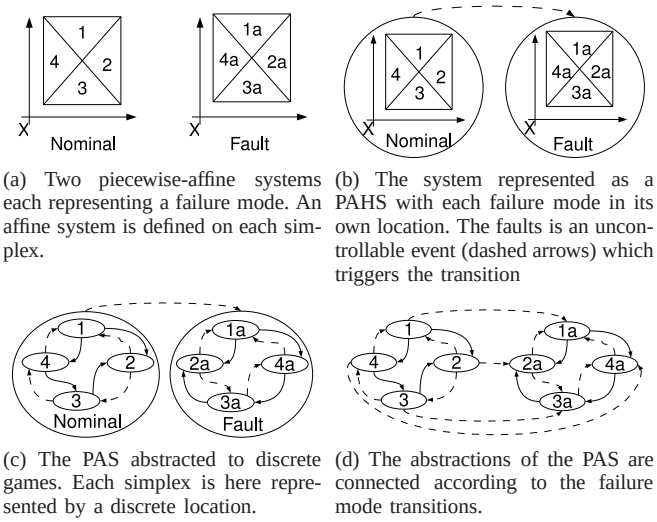
For controllability it must also be possible to find another set of inputs w_j at the same vertices where the derivative points out of the hull of the simplex and which ensures that any fixed point is outside the simplex.

B. Discrete Game Definition

The discrete game abstraction used in this paper is inspired by [19], [20] and is defined as $D = \{Q, Q_0, W, E, T, c\}$, where Q is a set of locations, $Q_0 \in 2^Q$ is a set of initial locations, E is a set of events that can be either controllable or uncontrollable, $W : Q \times E \to Q$ is a set of transitions connecting locations, $T : Q \times E \to Q$ is a special set of transitions triggered from external events which is inherited from the PAHS and c is a monotonically increasing clock. The goal of the game is derived from the requirements on the original PAHS and is specified using path invariants. This is discussed in more detail in section II-C.2.

Transitions are either controllable or uncontrollable; uncontrollable transitions can be taken by the environment (or "opponent") and has priority over controllable transitions. That is the environment always moves first.

Locations are divided into two categories: *committed* and *normal* locations. Normal locations are abstractions (discrete equivalents) of a simplex in the state space whereas committed locations represents controller choices. No time can pass in a committed location whereas a normal location must be left in finite time unless all outgoing transitions belonging to W are controllable.



(a) Two piecewise-affine systems each representing a failure mode. An affine system is defined on each simplex.

(b) The system represented as a PAHS with each failure mode in its own location. The faults is an uncontrollable event (dashed arrows) which triggers the transition

(c) The PAS abstracted to discrete games. Each simplex is here represented by a discrete location.

(d) The abstractions of the PAS are connected according to the failure mode transitions.

Fig. 1: Some essential steps for the FTC design procedure. The last step shown is the discrete game abstraction for which winning strategies are to be found.

C. Finding Winning Strategies

The goal of this method is to be able to find winning strategies to the discrete game abstraction and using the methods from [16] to refine the result to control laws on the individual simplices of the PAHS. A winning strategy will ensure that no matter which moves the environment takes (which faults occur) the controller will be able to meet the requirements.

The clock, c, is used to ensure progress of the game and that controller decisions in the committed locations is taken immediately. Because of this clock the discrete game abstraction is in reality a timed game and any method used to solve the game needs to be able to handle timed games or it must be possible to represent the effects of the time constraints in some way. This precludes using tools such as TCT and Supremica [21], [22] as they have no way of representing eventuality. Instead UppAal Tiga [23] is chosen as it can analyse properties and synthesise controllers for a quite general class of timed games.

Given a discrete abstraction the procedure now consists of the 3 following steps.

1) Converting the Discrete Game to UppAal Syntax: The syntactical conversion of the discrete abstraction to an UppAal representation of a timed game is quite straight forward. All locations, transitions and clocks are converted directly to the equivalent objects of UppAal.

To ensure that the semantics of the UppAal timed game are the same as those defined for the discrete abstraction it is necessary to add time invariants to the locations and guards to the transitions as follows:

1) Add invariant $c \leq 1$ to all normal locations where at least one outgoing W-transition is uncontrollable.
2) Add guard $c < 1$ to all outgoing controllable W-transitions from a normal location.
3) Add guard $c =< 1$ to all outgoing uncontrollable W-transitions from a normal location.
4) Add action $c = 0$ to all incoming transitions to a normal location.

The location invariant, *1)*, ensures that only finite time is spent in the location and the action on incoming transitions, *4)*, ensures that clock is always reset when entering a normal state. The guards, *2)* and *3)*, ensure that no deadlocks occur if the controller chooses to block all the controllable transitions by forcing the environment to choose one of the uncontrollable outgoing transitions.

Lastly it is necessary to take special care of converting the initial set as there can only be one initial location in the UppAal syntax. This can be solved by adding an extra location with transitions to all the states in the initial location set Q_0. If these transitions are all uncontrollable the corresponding controller must be able to fulfill the requirements specifications regardless of which of the possible initial states the system starts in. The transitions can also be made controllable to investigate whether it is possible to find an initial location from which the requirements specifications can be held.

2) Converting Requirement Specifications: The usual type of controller requirement specifications in the frequency or time domains can not directly be used for PAHS controller design based on discrete game abstractions. As the continuous dynamics is being abstracted away it is only possible to describe requirements in terms of locations to reach, stay in or avoid. The specification is thus done in the state (phase) space with the tessellation acting as a quantisation. The possible requirements that can be described using the discrete abstraction are therefore limited by the tessellation so the requirements specifications hence has to be considered during the creation of the discrete abstraction.

An example of the type of specifications that can be expressed is shown in Fig. 2, which shows a desired velocity profile in a 2-dimensional velocity/position state space. Notice that the profile has been encapsulated by simplices that cover the desired profile plus the acceptable error bounding box. The goal of the control is to start in the initial set marked with I and to reach the goal set marked G while staying within the error bounds, shown with dashed lines.

Notice that the specifications are absolute e.g. it is not possible to specify a maximum of 5% overshoot without specifying the step size. This is due to the specifications being moved to state space and the hybrid behaviour of the underlying PAHS. For fault tolerant control it is natural that faults can only be recovered from in certain areas of the state space and this is reflected nicely in this type of requirements specification.

In UppAal the requirements are specified using a subset of timed computational tree logic, T-CTL [20]. The basic property of CTL is that it can specify execution traces such as "the system will eventually reach a location *G* without visiting any states in *X*". UppAal is then able to verify if such a statement is true, and a control strategy ensuring the validity of the CTL-statement can be computed if it exists.

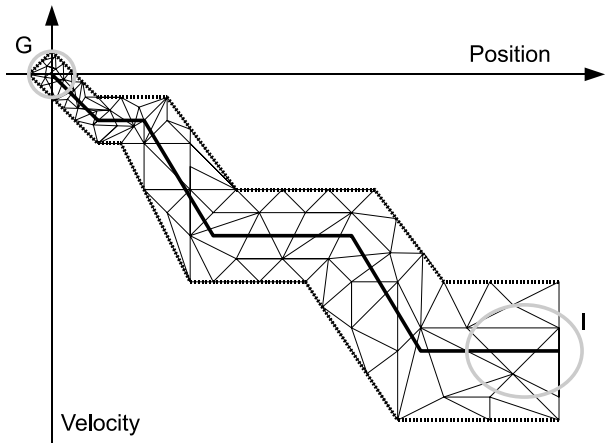


Fig. 2: Tessellated velocity profile in a 2-dimensional state space, with initial location set marked as I and goal set as G.

Table I shows a number of UppAal CTL specifications relevant for use with PAHS control.

CTL-expression	Specification
E[] not X	A path exist potentially relying on uncontrollable transitions that avoids state *X*.
A[] not X	All paths avoids *X*.
Control: A<> G	A control strategy exists such that *G* is eventually reached
Control: A[] not X	A control strategy exists such that all paths avoids *X*
Control: A[] not (X or Y)	A control strategy exists such that all paths avoids *X* and *Y*
Control: A[not X U G]	A control strategy exists such that all paths avoids *X* until the goal state *G* is reached.

TABLE I: CTL specifications relevant for PAHS control

By combining these expressions it is possible to specify that the state should reach a given set of simplices, *G*, while avoiding others, *X*. For the rendezvous example in Fig. 2 the CTL expression to generate the controller would look something like; *Control: A[not X U G]* for the initial set I and *Control: A[] G* for the initial set G, with X being the set of simplices outside the error bound. The specification has to be divided into two as UppAal does not support nested CTL expressions.

The rendezvous example can be expanded to include faults. To represent actuator faults a PAS could be added with an identical state space tessellation but with the discrete equivalents computed with respect to the fault mode dynamics. The two fault modes can be linked as described in section II-A and the goal set, *G*, should be a union of the goal set for the nominal mode and for the fault mode. The controller generation would then proceed exactly as shown above yielding a fault tolerant control strategy.

D. The Sliding Mode Problem

During analysis of the properties of controllers generated using the discrete game abstraction method it was discovered that it is possible for the controlled system to exhibit sliding mode behaviour.

Specifically for PAS this can occur on the boundary between two simplices, i.e. on a facet. This behaviour can have undesirable effects such as the state leaving a set of simplices through a vertex. The ability of the discrete game abstractions to accurately capture the dynamics of a PAS can thus be compromised and sliding modes must thus be avoided. The formal conditions and solutions to the sliding mode problem is out side the scope of this paper. Instead we present a simple UppAal workaround found as part of this research.

The UppAal workaround assures that no sliding modes will occur on codimensional[1]-1 facets e.g. lines when $n = 2$ and triangles for $n = 3$. Facets with codimension greater than one are very hard to hit as they have measure 0 and are thus not considered.

One sufficient condition to avoid sliding is not to reenter a simplex that was just left. Intuitively this makes sense as it precludes the system from switching multiple times between two simplices which is a behaviour usually required for sliding modes when implemented on real/simulated systems.

The requirement to not reenter a simplex can be implemented in UppAal adding an observer. The observer has two locations; one initial location indicating that no violation of the constraint has occurred, *obs*. *init* and one location indicating that a violation has occurred, *obs*. *violated* . The transition between these two locations is guarded by a signal, *h*, which is broadcast once the violation has been detected. To detect the violation a bounded integer variable, $r \in [0, \ldots, R_{max}]$, is added to the UppAal model where R_{max} is the number of simplices in the original PAHS. On exit from a location representing a simplex, Δ_i, r is assigned the index i. On each transition into a location representing a simplex, Δ_j, if $r = j$ then the constraint has been violated and the signal *h* is broadcast.

To ensure that control strategies found with UppAal do not violate this constraint the CTL specification has to be amended such that the set of states to be avoided *X*, see table I, *X*=*X*∪*obs*. *violated*

E. Refinement of the Winning Strategy

The strategy obtained from UppAal contains a choice of discrete equivalent for each committed location and an action for each reachable normal location. The action is on the form of either blocking all possible transitions or trying to take a specific transition.

These actions can be transformed in to continuous control laws on the individual simplices using the refinement tools in the *PAHSCTRL* toolbox [17]. For each simplex the choice of discrete equivalent determines which facets to block and the transition chosen in the winning strategy is directly translated to a desired exit facet which is then unblocked. The resulting control law, as shown in fig 3, is a catalogue of controllers, one for each simplex, $u_i = K_i x + g_i$, leading to a piecewise-affine control law. It is assumed that an observer exists which can detect failures (FDI) and notify the controller when a facet is crossed. Furthermore the designed controllers require full state observability so the controller design relies on the super-positioning principle i.e. that the controllers can be designed independently of the observers. This limits the controller to handle faults in the plant while the observer must handle any faults in the sensory subsystem.

[1]A facet of codimension-k belongs to a subspace of L of dimension $n-k$

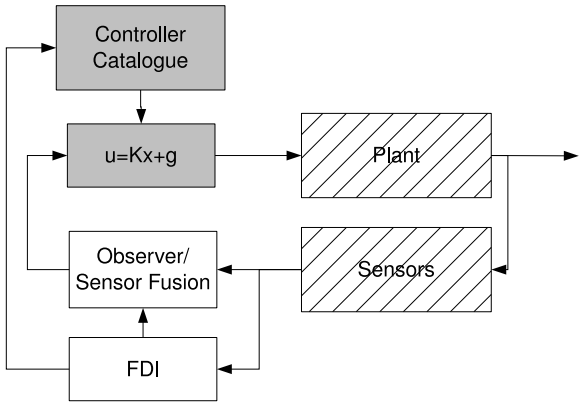


Fig. 3: The controller structure resulting from the refinement procedure. Grey boxes are the controller, white boxes the FDI and observers and hatched boxes represent the physical plant and sensor systems.

III. Example

To show the feasibility of the method a simple double integrator example with two modes is studied (1).

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

The two modes are the nominal mode and a failure mode, where the fault considered is an actuator fault. In the nominal mode $u \in [2; -2]$ and for the fault mode $u \in [2; -1]$, whereas the dynamics remain the same.

A. Obtaining the Discrete Abstraction

The first step in obtaining a discrete abstraction of the system is to get the system on the form of a PAHS. The state space of the two modes is partitioned using the simplicial complex shown in fig 5 and the resulting PAS are connected via an uncontrollable transition from the nominal mode to the failure mode, similar to Fig. 1b. Since the fault can happen at any time there is no guard on the transition and the reset map is the identity map as the actuator fault does not affect the state.

Figure 5 also shows the requirements used in this example. The goal of the game is starting from the light grey simplex, to reach and stay in one of the two grey simplices while avoiding the black simplices. Notice that in the failure mode there are fewer simplices to avoid, i.e. the requirements are relaxed if the fault occurs such that the controller has more

freedom. Note this specification has no relation to a real problem but is designed to show some of the capabilities of the method.

The model is implemented in Matlab and a discrete abstraction is found using the *PAHSCTRL* toolbox [17]. In Fig. 4 the discrete abstraction is shown as it looks after conversion to UppAal. As can be seen there are many states with each mode having 18 simplices and each simplex potentially having multiple discrete equivalents. This is a reflection of the computational complexity of the discrete abstraction step, which is worst case exponential with the number of continuous states. This is somewhat mitigated as the computation of discrete equivalents of the simplices are independent and it is thus possible to perform these computations in parallel.

Notice that there are fewer discrete equivalents per state in the failure mode, which reflects that the input space is smaller and thus there are fewer possible control actions.

B. UppAal Setup

As described in section II-C.2 it is necessary to specify two discrete games and combine the results if the requirement is to reach a goal set and stay there. Below are the two CTL specifications used in this example.

```
/* Spec 1: CTL specification to get from
     initial state to goal set */
control: A[not (obs.violated || DA.black
     || DA.X) U DA.goal]

/* Spec 2: CTL specification to stay in
    the goal set */
control: A[] (not (obs.violated ||
    DA.black || DA.X)) && (DA.start ||
    DA.goal)
```

DA.black is the unpartitioned state space, *DA.goal* is the goal set, *DA.X* is the set of forbidden simplices e.g. *DA.X = DA.c_1_9 || DA.c_1_17 || DA.c_1_14 || DA.c_2_14*.

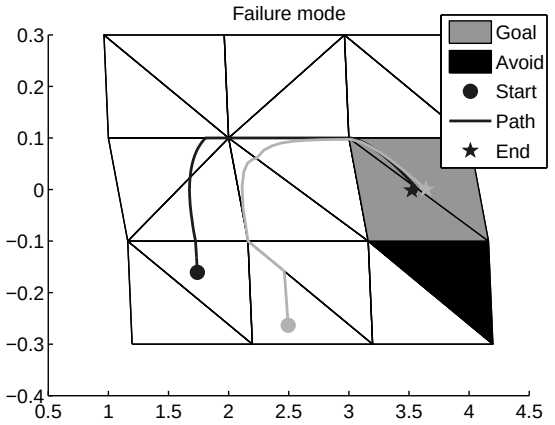
For each of the CTL specifications a solution is found and the resulting control laws are merged. This is possible as the first specification uses the initial set given in the requirements while the second specification uses the goal set as the initial set. The first specification brings the system to the goal set and the second ensures that once the goal set is reached that the state stays in it.

Solutions to both specifications are found by UppAal, and it is thus possible to generate controllers fulfilling the requirements. It should be noted that if the requirements are not relaxed in the failure mode then no solution to the game exists.

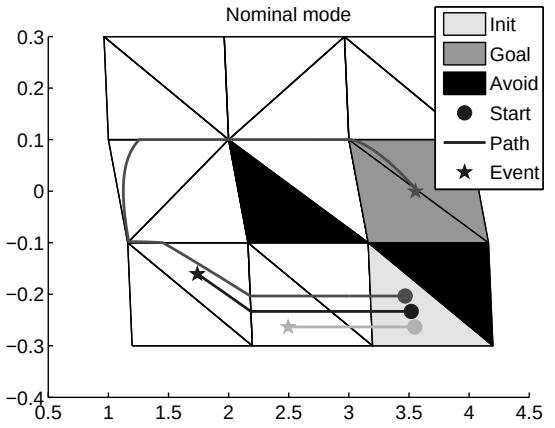
With the solutions found they can be converted into a controller catalogue using the *PAHSCTRL* toolbox.

C. Simulation

Three simulations of the controlled system has been performed using the *trap_101* ODE solver [24], which is especially designed for hybrid and switching systems. In the first simulation the fault occurs at $t = 4s$, in the second simulation the fault occurs at $t = 8s$ and in the third simulation no faults occurs. Each simulation is run for $50s$ and the controller catalogue for each run are identical. In Fig. 5 the three simulations are shown as a path in the nominal mode jumping to the failure mode when a fault occurs. The behaviour of the controlled system can be seen to vary greatly depending on when and if the fault occurs, which is mainly due to the relaxed requirements when the system is in the failure mode. This is particularly evident in the simulation with the fault at $t = 4s$ where the controller takes the opportunity to take a "short cut" through a simplex which must be avoided in the nominal mode.



(a) Traces of the simulation in the failure mode. The fault only occurs in two of the simulations and the third simulation thus leaves no trace in the failure mode.



(b) Traces of the simulation in nominal mode.

Fig. 5: Three simulations of the controlled system, two of them with faults occuring.

IV. Conclusion

In this paper, a method for automatic generation of fault tolerant controllers was given. The design procedure is based on finding winning strategies for discrete abstractions of a PAHSs using UppAal. This involves converting the discrete abstraction to an UppAal timed game and constructing a corresponding CTL specification based on the controller

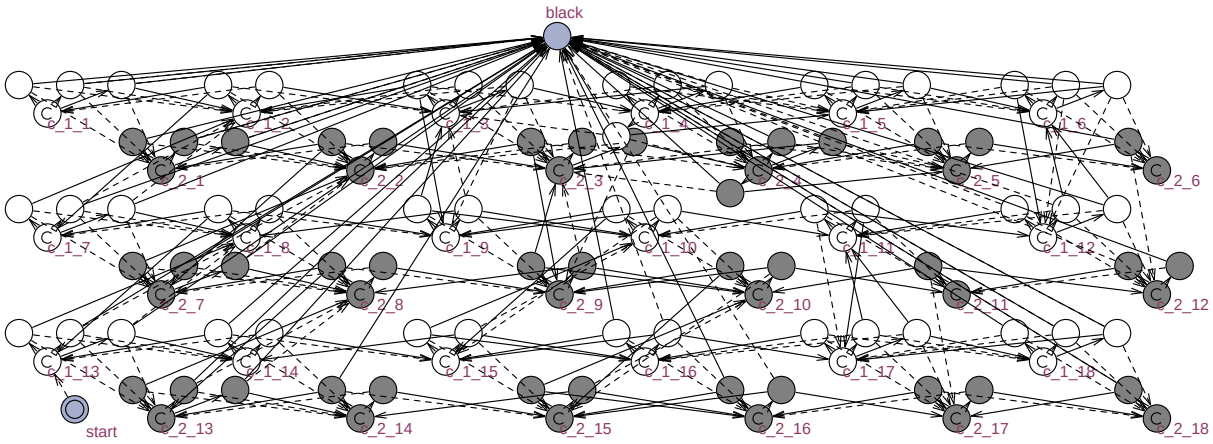


Fig. 4: Discrete game abstraction of the double integrator example. White locations belong to the nominal mode and grey locations belong to the failure mode. Locations marked with c are committed states, one for each simplex.

requirements. By parsing the UppAal output in Matlab and using the refinement procedures in the *PAHSCTRL* toolbox a catalogue of continues time controllers along with a corresponding switching strategy can be found. This was all illustrated by simulations of an automatically generated controller for a double integrator with a possible actuator failure.

When implementing controllers based on the discrete abstraction method sliding modes can occur on facets shared by simplices, and to combat this behaviour a UppAal workaround based on a location observer was presented. The formal conditions for solutions to the sliding behaviour are currently being investigated.

In summary the results confirm the feasibility of automating fault tolerant controller design. It is hoped that this work can serve as the basis for further automation of controller design, but there are still many challenges remaining, especially concerning methods for partitioning of the state space and the related specifications, optimal control and robustness with respect to noise and modelling errors.

REFERENCES

[1] M. Blanke, M. Kinnaert, J. Lunze, M. Staroswiecki, and J. Schröder, *Diagnosis and Fault-Tolerant Control*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[2] R. Iserman, *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer-Verlag, 2005.

[3] H. Niemann and J. Stoustrup, "Passive fault tolerant control of a double inverted pendulum - a case study," *Control Engineering Practice*, vol. 13(8), pp. 1047–1059, Aug. 2005.

[4] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. Courier Dover Publications, 1965.

[5] M. Mahmoud, J. Jiang, and Y. Zhang, *Active Fault Tolerant Control Systems: Stochastic Analysis and Synthesis*, ser. Lecture Notes in Control and Information Sciences. Springer, 2003, vol. 287.

[6] R. J. Patton and J. Chen, *Robust Model-Based Fault Diagnosis for Dynamic Systems*. Kluwer Academic Publishers, 1999.

[7] C. Hajiyev and F. Caliskan, *Fault Diagnosis and Reconfiguration in Flight Control Systems*. Kluwer Academic Publishers, 2003.

[8] G. Tao, S. Chen, X. Tang, and S. M. Joshi, *Adaptive Control of Systems with Actuator Failures*. Springer, 2004.

[9] L. C. G. J. M. Habets, P. J. Collins, and J. H. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Transactions on Automatic Control*, vol. 51, pp. 938–948, 2006.

[10] R. Wisniewski and J. A. Larsen, "Combinatorial vector fields for piecewise affine control systems," in *Proc. of the 17th IFAC World Congress*, 2008.

[11] L. Rodrigues and J. P. How, "Synthesis of piecewise-affine controllers for stabilization of nonlinear systems," in *42nd IEEE Conference on Decision and Control*, 2003, pp. 2071–2076.

[12] L. Rodrigues and J. How, "Automated control design for a piecewise-affine approximation of a class of nonlinear systems," in *IEEE American Cotrol Conference*, 2001, pp. 3189–3194.

[13] S. LeBel, L. Rodrigues, and A. Ng., "Piecewise-affine controller synthesis for a model of 2d orbital path following," in *IEEE Conference on Control Applications*, 2005, pp. 571–576.

[14] J. D. Grunnet, T. Bak, J. D. Bendtsen, and J. A. Larsen, "Discrete game abstraction for fault tolerant control synthesis," in *Proc. of IEEE CACSD '08*, 2008.

[15] L. Habets and J. H. van Schuppen, "Control to facet problems for affine systems on simplices and polytopes - with applications to control of hybrid systems," in *Proc. 44th IEEE CDC*, 2005.

[16] J. D. Grunnet, T. Bak, and J. D. Bendtsen, "PAHSCTRL - a control synthesis toolboox for piecewise-affine hybrid systems," in *Proc. of ECC '09*, 2009, accepted.

[17] J. D. Grunnet, "PAHSCTRL - a matlab toolbox for control of piecewise-affine hybrid systems," 2009. [Online]. Available: http://www.control.aau.dk/~grunnet/pahsctrl

[18] J. D. Grunnet, J. A. Larsen, T. Bak, and R. Wisnievski, "A piecewise affine hybrid systems approach to fault tolerant satellite formation control," in *Proc. of the 3rd International Symposium on Formation Flying, Missions and Technologies*, 2008.

[19] T. A. Henzinger, "The theory of hybrid automata," in *Proc. of the 11th Annual IEEE Symposium on Logic in Computer Science*, 1996.

[20] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, pp. 134–152, 2004.

[21] L. Feng and W. Wonham, "TCT: A computation tool for supervisory control synthesis," in *Proc. of the 8th International Workshop on Discrete Event Systems*, 2006.

[22] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proc. of the 8th International Workshop on Discrete Event Systems*, 2006.

[23] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime, *CONCUR 2005 - Concurrency Theory*. Springer Berlin / Heidelberg, 2005, ch. Efficient On-the-Fly Algorithms for the Analysis of Timed Games, pp. 66–80.

[24] J. H. T. . D. Kebede, "Modeling and simulation of hybrid systems in matlab," in *Proc. of IFAC World Congress, SF, 1996*, 1996.