2019 IEEE 58th Conference on Decision and Control (CDC)
Palais des Congrès et des Expositions Nice Acropolis
Nice, France, December 11-13, 2019

Incremental Abstraction Computation for Symbolic Controller Synthesis in a Changing Environment

Yunjun Bai*, Kaushik Mallik*, Anne-Kathrin Schmuck, Damien Zufferey, and Rupak Majumdar

***Abstract*— Abstraction-Based Controller Synthesis (ABCS) is an emerging field for automatic synthesis of correct-by-design controllers for non-linear dynamical systems in the presence of bounded disturbances. A major drawback of existing ABCS techniques is the lack of flexibility against changes in the disturbance model; any change in the model results in a complete re-computation of the abstraction and the controller. This flexibility is relevant to situations when disturbances are learned or estimated during operation in an environment which is previously not known precisely. As time passes, the disturbance model is progressively refined. The monolithic nature and high computational cost of existing algorithms make ABCS unsuited for such scenarios.**

In this paper, we present an incremental algorithm to locally adapt abstractions to changes in the disturbance model. Only the parts of the space which are affected by the changes are updated and the rest of the abstraction is reused. Our new abstraction method allows to apply existing incremental techniques to update the discrete controller locally for the changed abstraction. This results in an incremental ABCS algorithm. We empirically show the benefit of dynamic abstraction adaptation on two large examples: a 5-dimensional vehicle model and a 12-dimensional quadrotor model. In both cases, the speed-up over complete re-computation is significant.

I. Introduction

Abstraction-based controller synthesis (ABCS) is a fully automated model-based controller synthesis technique for non-linear, continuous dynamical systems with respect to temporal control objectives [22], [4], [10], [19], [17], [12]. In ABCS, a model of the non-linear dynamical system, under sampled-time semantics, is abstracted to a finite-state, two-players game. Using automata-theoretic algorithms for reactive synthesis [15], one can automatically compute an abstract controller which can be refined to a continuous controller for the original system. The correctness of the technique depends on the existence of an alternating refinement or feedback refinement relation [1], [14], [17], [19] between the original system and its abstraction.

Like any other model-based controller synthesis techniques, the formal guarantees of ABCS rely on the assumption of an accurate system model. The system model is typically of the form

$$\dot{x} \in f(x,u) + [\![-d, d]\!]$$

where x is the state vector, u is the control input, f is a function representing the nominal (unperturbed) dynamic model of the system, and d is a vector with non-negative elements representing the perturbation bound. The perturbation bound captures the aggregate of both modeling inaccuracies and the environmental disturbances. It is because the exact system model is of such central importance that, in safety critical applications, the system designers are forced to use the most pessimistic bound on d to capture the effect of any unforeseen deviation of the model that might appear in the future. This often leads to an excessively restrictive controller which fails to suit any practical purpose.

Instead, we would be better off starting with a more optimistic perturbation model and, then, update the controller when changes happen. Unfortunately, whenever there is any change in the model, no matter how small, the abstraction and the controller need to be computed from scratch. This is wasteful and impractical especially when the changes only affect a small part of the system model's state space.

In this paper, we present an adaptation procedure of system abstractions to local changes in the perturbation bound d of the system model. For this purpose, we first switch to a slightly more general form of the system model admitting a state and input-dependent perturbation bound:

$$\dot{x} \in f(x,u) + [\![-d(x,u), d(x,u)]\!],$$

where for any given control input u, $d(\cdot, u)$ is a continuous function. We assume that the nominal dynamics f remain fixed. Then given an initial abstraction $\mathcal{A}$ of the system, and a new perturbation bound $d'(\cdot,\cdot)$ with the property that the value of $d(\cdot,\cdot)$ and $d'(\cdot,\cdot)$ differ only on a known subset of their arguments, we show how an abstraction $\mathcal{A}'$ of $\dot{x} \in f(x,u) + [\![-d'(x,u), d'(x,u)]\!]$ can be obtained, by changing the abstraction $\mathcal{A}$ as little as possible. The resulting abstraction allows to apply existing incremental techniques for dynamic synthesis over finite state game graphs to update the existing discrete controller locally [7]. This would result in an incremental ABCS algorithm.

To explain our proposed adaptation procedure for system abstraction, we first recall how abstractions are typically computed. A simple but effective abstraction procedure first partitions the state and input spaces using disjoint hypercubes to form the abstract state space. Then, a non-deterministic transition relation is computed over this abstraction. The transition relation provides, for each state-input pair, the possible states that can be reached under the effect of the dynamics in the presence of disturbances. This approach has been implemented in several tools [16], [20], [12], [13].

Yunjun Bai is with State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing, China, 100190. baiyj@ios.ac.cn, Kaushik Mallik, Anne-Kathrin Schmuck, Damien Zufferey, and Rupak Majumdar are with MPI-SWS, Germany. {kmallik, akschmuck, zufferey, rupak}@mpi-sws.org

*The first two authors have equal contribution.

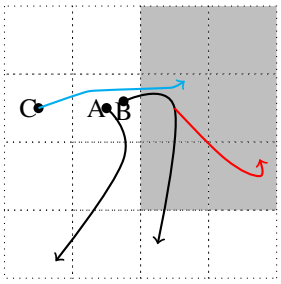


Fig. 1. Each square represents an abstract state. The grey regions form $\widehat{R}$. The lines represent trajectories in $[0, \tau]$. A nominal trajectory starting at A may not pass through $\widehat{R}$, but a perturbed trajectory starting at B may, and the new disturbance may give rise to the red trajectory, which gives rise to a new transition in the abstraction. The updated disturbance may also invalidate a previous trajectory, removing an abstract transition.

Essentially, adapting the abstraction requires a partial re-computation of the abstract transitions for the state-input pairs affected by the changes in the disturbance model. Identifying such state-input pairs is the core of our algorithm. Assume that the disturbance has changed in a region called $\widehat{R}$. The challenge is that even though we precisely know the region $\widehat{R}$, there might be other states around $\widehat{R}$ from where trajectories could enter $\widehat{R}$. Even more challenging is to address those trajectories which only enter $\widehat{R}$ from outside during the inter-sampling period, and leave $\widehat{R}$ before the next sampling takes place (see Fig. 1). We detect these cases by peeking in the intermediate computations of the numerical solver, and marking all the intermediate state-input pairs from where trajectories might have entered $\widehat{R}$. Our algorithm runs in time proportional to the size of the region for which the disturbance model changes, inflated by a term depending on the speed of the unperturbed dynamics. In many typical scenarios, if the region of change is small compared to the entire state space, the dynamic algorithm can be significantly faster than a complete re-computation.

In the following, we motivate our problem and informally present our approach using an example of a robotic vehicle in a not precisely known environment.

Motivating Example

Consider the example of a vehicle inside a building which tries to reach the location B starting from location A while avoiding obstacles. Let us assume that the vehicle only has a rough estimate of the environment in the beginning, and as it moves around the building, it gathers more and more environmental data using its sensors. Any new information about the environment can then be used to update the abstraction and the controller. Fig. 2 captures three different phases of the controller under changing environment data: (a) corresponds to the initial controller, and (b) and (c) represent two subsequent updates.

As can be seen in Fig. 2(a), the vehicle has to pass through two rooms R_1 and R_2, and either of the two corridors C_1 and C_2 in order to reach B. Assume that the disturbance values in R_1, R_2, C_1 and C_2 are given by the vectors w_1, w_2, w_1 and w_3 respectively, where $w_1 < w_2 < w_3$ (element-wise). Initially, the disturbance is set to w_1 everywhere (Fig. 2(a)). We start by synthesizing a symbolic controller off-line using ABCS. During the closed loop operation/simulation phase, when the vehicles reaches near R_2, it gathers that the actual disturbance in R_2 is w_2 (the hatched part marks a change in disturbance). Instead of an expensive complete re-computation of the abstraction, we only update the transitions in and around R_2. As the transitions may change, the controllable region of the state space might change as well (Fig. 2(b)). The green region in the state space is the controller domain (set of winning states in synthesis), and the purple region is outside the controller domain (set of losing states in synthesis). For instance, the narrow passage above the obstacle in R_2 is no longer passable due to the increased disturbances.

The second re-computation takes place when the vehicle is close to C_2, when the sensor data suggests that the actual disturbance in C_2 is w_3 (cross-hatched in Fig. 2(c)). Now w_3 is so high that the whole corridor C_2 becomes inaccessible, and the updated controller makes the vehicle take the corridor C_1 (see Fig. 2(c)).

For a 3-dimensional vehicle model [19], and for disturbance values $w_1 = (0.01, 0.01, 0.01)$, $w_2 = (0.033, 0.033, 0.033)$, $w_3 = (0.07, 0.06, 0.07)$, the arrow-labels in Fig. 2 depict comparisons between the time needed for the local incremental updates of the abstraction and the global complete re-computations of the same. It is observed that the local update is approximately $3\times$ faster during the first re-computation, and a whopping $12\times$ faster during the second re-computation. It is evident that the smaller the area of disturbance update, the greater is the speedup.

Related Work

Adaptability of controllers with changing system model while providing formal guarantees has been explored in several recent papers. One group of results assume that the system behavior is accurately known in the form of a parametric model over a known parameter space [21], [18]. An adaptive control scheme then synthesizes a control policy off-line, which can adapt to all possible changes in parameter values. However, analyzing the effects of all parameters puts a high computational burden on such techniques.

The second group of results is based on the idea of learning the environment from sensor data, while remaining under the supervision of a safe controller [9], [2], [5]. Our work is inspired by this line of research, and we intend to extend this in the context of ABCS. This paper can be viewed as a first step in that direction. The biggest advantage of ABCS is that it can handle at the same time a wide variety of nonlinear system models and a very expressive class of control specifications, e.g. linear temporal logic.

In this paper, we keep open the question of how to ensure that the present system state is still winning after a controller re-computation, and implicitly assume that all the trajectories can be safely traced backwards always [11], [8]. We also assume that the information of the disturbance values is

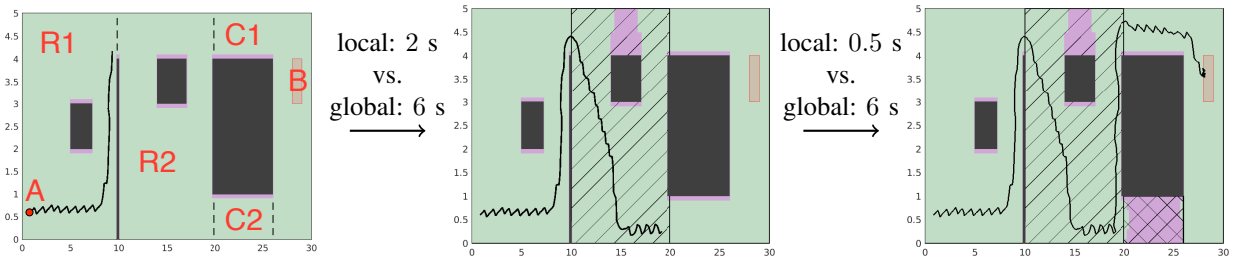


Fig. 2. Reach-avoid problem for a vehicle to reach the target (B) from the initial position (A) while avoiding obstacles (black) and rejecting dynamic disturbances. The hatched region and the cross-hatched region have disturbance values w_2 and w_3 respectively, while the remaining work space has disturbance value w_1 ($w_1 < w_2 < w_3$ element-wise). Initially, ABCS is performed while assuming a disturbance value of w_1 in the overall work space. The resulting (partial) closed-loop trajectory is shown in (a). During deployment, actual disturbance values are observed, the abstraction is updated and the controller is re-synthesized. The resulting (partial) trajectories for updates w.r.t. w_2 and w_3 are depicted in (b) and (c), respectively. The controller domain is indicated in green, with the violet areas indicating its complement. The controller domain reduces during re-computation, due to observed increases in the disturbance value. Computation times for local updates (top) and global re-computation (bottom) of the abstraction are indicated between sub-figures.

available to the system, either through direct measurement or indirect means.

II. Preliminaries

We recall the setting of ABCS.

A. Systems and Refinement

A *system* $S = (X, U, F)$ consists of a state space X, an input space U, and a transition relation $F \subseteq X \times U \times X$. A system is called *finite* if X and U are finite sets. A *run* of a system is a finite or infinite sequence $\xi = x_0 x_1 \ldots$ such that for each $i \in dom(\xi)$ there is an $u_i \in U$ such that $(x_i, u_i, x_{i+1}) \in F$. The set of runs of S is denoted $\mathcal{B}(S)$ and the set of runs of S starting from a state $x \in X$ is denoted $\mathcal{B}(S)(x)$. A *controller* $C : X \to U$ is a function mapping states to inputs which restricts the set of behaviors of a system: a run $\xi = x_0 x_1 \ldots$ is *compatible* with C if for each i, we have $(x_i, C(x_i), x_{i+1}) \in F$.

The goal of abstraction-based controller synthesis is to solve the problem on an abstract system and then refine a controller to the original system. For this to work, we need a suitable notion of abstraction. Feedback refinement relations provide such a notion [1], [19].

Definition 1 (Feedback refinement relations): Let $S_i = (X_i, U_i, F_i)$ for $i \in \{1, 2\}$ be two systems s.t. $U_2 \subseteq U_1$. Let $\pi_i : X_i \to 2^{U_i}$ be the mapping $\pi_i(x) = \{u \in U_i \mid \exists x' \in X_i \cdot (x, u, x') \in F_i\}$ giving the set of allowed inputs in state x of S_i. A *feedback refinement relation* from S_1 to S_2 is a relation $Q \subseteq X_1 \times X_2$ such that for all $x_1 \in X_1$ there is a unique $x_2 \in X_2$ such that $(x_1, x_2) \in Q$, and for all $(x_1, x_2) \in Q$, the following holds:

(i) $\pi_2(x_2) \subseteq \pi_1(x_1)$ and
(ii) for all $u \in \pi_2(x_2)$, for all $x_1' \in X_1$ such that $(x_1, u, x_1') \in F_1$ there is an $x_2' \in X_2$ such that $(x_2, u, x_2') \in F_2$ and $(x_1', x_2') \in Q$.

We use $S_1 \preccurlyeq_Q S_2$ to represent that Q from S_1 to S_2 is a feedback refinement relation. We say S_2 is an abstraction of S_1 if $S_1 \preccurlyeq_Q S_2$. It is well known that $S_1 \preccurlyeq_Q S_2$ implies that any controller $C_2 : X_2 \to U_2$ solving a control problem on S_2 can be refined to a controller $C_1 : X_1 \to U_1$ solving the control problem on S_1 [1].

B. Time-Sampled Control Systems

We consider continuous-time *control systems* $\Sigma = (X, U, d, f)$, where $X = \mathbb{R}^n$ is the state space, $U \subset \mathbb{R}^m$ is the input space, $d : X \times U \to X$ is a function continuous in X modeling the upper-bound of the state dependent disturbances, and $f : X \times U \to X$ is a locally lipschitz continuous function for all $u \in U$ modeling the unperturbed system dynamics. The overall dynamics is expressed by the following differential inclusion:

$$\dot{x} \in f(x, u) + [\![-d(x, u), d(x, u)]\!]. \tag{1}$$

Since the set of disturbances $[\![-d(x, u), d(x, u)]\!]$ at any given state $x \in X$ and input $u \in U$ is symmetric about 0^n, hence w.l.o.g. we assume that $d(x, u)$ is positive for all $x \in X$ and input $u \in U$. We assume that f is explicitly known and is continuously differentiable in x for all $u \in U$. The set $[\![-d(x, u), d(x, u)]\!]$ represents the set of all possible disturbances capturing the unmodeled dynamics and environmental uncertainties. Given an initial state $x_0 \in X$, a constant control input[1] $u \in U$ for time $t_f \in \mathbb{R}_+$ and any interval $I \subset [0, t_f]$, a possible evolution of Σ in I is given by any continuous trajectory $\xi_u : I \to X$ which satisfies $\dot{\xi}_u(t) \in f(\xi_u(t), u) + [\![-d(\xi_u(t), u), d(\xi_u(t), u)]\!]$. Moreover we define the *nominal trajectory* $\varphi_{x_0 u} : I \to X$, which satisfies the unperturbed differential equation $\dot{\varphi}_{x_0 u}(t) = f(\varphi_{x_0 u}(t), u)$ and $\varphi_{x_0 u}(0) = x_0$.

Given a fixed time sampling parameter $\tau > 0$, we define a system $\Sigma_\tau = (X, U, f_\tau)$ that represents the *sampled-time representation* of a control system Σ, where $f_\tau \subseteq X \times U \times X$ is the transition relation s.t. for all $(x, u) \in X \times U$, we have $(x, u, x') \in f_\tau$ iff there is a solution of (1) for the constant input u s.t. $\xi(0) = x$ and $\xi(\tau) = x'$. The state space of Σ_τ is still infinite; our goal is to define a *finite* abstract system $\widehat{\Sigma}$ that is an abstraction of Σ_τ.

[1]We restrict our notation to piecewise constant control inputs as more general control inputs will be unnecessary for the later part.

III. Global Computation of Abstraction

Fix a control system $\Sigma = (X, U, d, f)$ and its sampled-time representation $\Sigma_\tau = (X, U, f_\tau)$ for a fixed sampling time $\tau > 0$. Our goal is to compute a finite-state abstraction $\widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f})$. We work in the usual setting of ABCS, where an abstraction is obtained by 1) partitioning the state space and input space of Σ_τ into finitely many hypercubes, and 2) over-approximating the transitions to obtain the desired finite-state abstraction $\widehat{\Sigma}$.

We consider abstract state spaces defined by hyper-rectangle covers. A *hyper-rectangle* $[\![a, b]\!]$ with $a, b \in (\mathbb{R} \cup \{\pm\infty\})^n$ defines the set $\{x \in \mathbb{R}^n \mid a_i \leq x_i \leq b_i \text{ for } i \in \{1, \ldots, n\}\}$; it is non-empty if $a < b$ (element-wise). For $\eta \in \mathbb{R}^n_{>0}$, we say that a hyper-rectangle $[\![a, b]\!]$ has diameter η if $|b - a| = \eta$, where $|\cdot|$ represents the element-wise absolute value. The *center* of a non-empty hyper-rectangle $[\![a, b]\!]$, written $ctr([\![a, b]\!])$, is the point $(\frac{1}{2}(b_1 - a_1), \frac{1}{2}(b_2 - a_2), \ldots, \frac{1}{2}(b_n - a_n))$. A set C of hyper-rectangles is called a *cover* of the state space X if every $x \in X$ belongs to some hyper-rectangle in C.

In the following, we make the standard assumption that there is a compact subset $X' \subseteq X$ of the state space which is of interest to the control problem. Given a discretization parameter $\eta \in \mathbb{R}^n_{>0}$, the abstract state space $\widehat{X}$ is a (finite) cover of the compact set X' with hyper-rectangles of diameter η together with a finite number of larger hyper-rectangles which cover $X \setminus X'$.

The abstract transition relation $\widehat{f}$ is defined as an over-approximation of the reachable states in a single time sampling period τ. The over-approximation is formalized by a *growth bound*.

Definition 2 (Growth bound): Let $\Sigma = (X, U, d, f)$ be a control system. Given a fixed sampling time $\tau > 0$ and sets $K \subset X$ and $U' \subset U$, a *growth bound* is a map $\beta_\tau : \mathbb{R}^n \times \mathbb{R}^n_{>0} \times U' \to \mathbb{R}^n_{>0}$ satisfying the following conditions:

(i) $\beta_\tau(p, r, u) \geq \beta_\tau(p, r', u)$ whenever $r \geq r'$ and $u \in U'$;

(ii) for all $p \in K$ and $u \in U'$, $[0, \tau] \subset dom\, \varphi_{pu}$, and if ξ_u is a trajectory of (1) on $[0, \tau]$ with $\xi_u(0) \in K$ then

$$| \xi_u(\tau) - \varphi_{pu}(\tau) | \leq \beta_\tau(p, | \xi_u(0) - p |, u) \quad (2)$$

holds component-wise. Recall that $\varphi_{pu}(t)$ denotes the nominal trajectory starting from p.

Def. 2 is an adaptation of the growth bound in [19]; the difference is that our growth bound also depends on the initial state of the nominal trajectory. The intuition behind this extra requirement is that the deviation of the actual trajectory from the nominal trajectory is no longer independent of the path of the trajectory when the disturbance is not uniform. This will be clear in the subsequent development.

We introduce ComputeGrowth as a black-box procedure for growth bound computation. (A possible implementation of ComputeGrowth can be found in [3], and is omitted here due to space constraints.) The inputs to the procedure ComputeGrowth are (i) the center of the current abstract state $ctr(\widehat{x})$ (i.e., the starting point of the nominal trajectory), (ii) the input $\widehat{u}$, (iii) the time discretization parameter τ, (iv) the state-space discretization parameter η, and, optionally, (v) some set of abstract states $\widehat{R} \subseteq \widehat{X}$. The outputs of ComputeGrowth are (i) $\varphi_{ctr(\widehat{x})\widehat{u}}(\tau)$, (ii) $\beta_\tau(ctr(\widehat{x}), \frac{1}{2}\eta, \widehat{u})$, and (iii) a boolean flag $isOverlap$ (when $\widehat{R}$ is specified) which is set to true iff there is $t \in [0, \tau]$ s.t. $[\varphi_{x\widehat{u}}(t) - r(t), \varphi_{x\widehat{u}}(t) + r(t)] \cap \widehat{R} \neq \emptyset$. The role of the set $\widehat{R}$ and the flag $isOverlap$[2] will be clear in Sec. IV. Note that additional assumptions on the function $d(\cdot, \cdot)$ might be required depending on how ComputeGrowth is computed.

Given the notion of state space partition and growth bound, we can now define the finite-state abstraction as follows:

Definition 3: Let $\Sigma = (X, U, d, f)$ be a control system, let $\tau > 0$ be a sampling time, $\eta \in \mathbb{R}^n_{>0}$ a discretization parameter, and $\beta_\tau : \mathbb{R}^n \times \mathbb{R}^n_{>0} \times \widehat{U} \to \mathbb{R}^n_{>0}$ a growth bound. A finite-state abstraction $\widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f})$ of Σ_τ consists of a finite set $\widehat{X}$ which forms a cover of X with hyper-rectangles of diameter η, a finite set $\widehat{U} \subseteq U$ of inputs, and a transition relation $\widehat{f} \subseteq \widehat{X} \times \widehat{U} \times \widehat{X}$ which satisfies, for all $\widehat{x}, \widehat{x}' \in \widehat{X}$ and $\widehat{u} \in \widehat{U}$, $(\widehat{x}, \widehat{u}, \widehat{x}') \in \widehat{f}$ iff

$$\big(\varphi_{ctr(\widehat{x})\widehat{u}}(\tau) + [\![-\gamma, \gamma]\!]\big) \cap \widehat{x}' \neq \emptyset, \quad (3)$$

where $\gamma = \beta_\tau(ctr(\widehat{x}), \frac{1}{2}\eta, \widehat{u})$.

The procedure called GlobalAbs presented in Alg. 1 summarizes the steps for computing the finite-state abstraction of a control system Σ for some given discretization parameters η and τ.

The following theorem is adapted from [19, Thm. VIII.4].

Theorem 1: For every control system $\Sigma = (X, U, d, F)$, for every sampling time τ, state discretization η, and growth bound β, we have $\Sigma_\tau \preccurlyeq_Q \widehat{\Sigma}$, through the feedback refinement relation $(x, \widehat{x}) \in Q$ iff $x \in \widehat{x}$.

IV. Local Re-computation of Abstraction

Consider the control system $\Sigma = (X, U, d, f)$ and assume that we have computed an abstraction $\widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f})$ using parameters τ and η. Now imagine that the model Σ is updated to $\Sigma' = (X, U, d', f)$, where d' is the new disturbance function. For example, a new model of the disturbance can be learnt over the course of operation or due to availability of better knowledge about the system. The existing abstraction $\widehat{\Sigma}$ may no longer be a sound abstraction of the updated model Σ'. One option is to recompute a new abstraction but this may be wasteful if, e.g., d' differs from d only in a small part of the state space. Our second main contribution is an algorithm that instead of computing the new abstraction $\widehat{\Sigma}'$ from scratch, makes minimal changes in $\widehat{\Sigma}$ to obtain $\widehat{\Sigma}'$.

We assume that we know a set R of states over-approximating the set of states and inputs where the functions d and d' differ:

$$\{(x, u) \in X \times U \mid d(x, u) \neq d'(x, u)\} \subseteq R. \quad (4)$$

[2]The feature of ComputeGrowth to support $\widehat{R}$ as input and provide $isOverlap$ as output is not there in the implementation in [3]. However it can be easily added in the internal ODE solver that $isOverlap$ uses.

Its abstract representation is the set

$$\widehat{R} := \{(\widehat{x}, \widehat{u}) \in \widehat{X} \times \widehat{U} \mid \exists x \in \widehat{x}.\, (x, \widehat{u}) \in R\}. \quad (5)$$

The main question that needs to be addressed is for which abstract states outgoing transitions need to be re-computed. Clearly, all transitions leaving states in $\widehat{R}$ may be updated. In addition, it is possible that trajectories originating from a neighboring abstract state to some state in $\widehat{R}$ pass through a state in $\widehat{R}$ (see Figure 1). Thus, such transitions may also be updated. Moreover, for any such updated transition, we also check its neighbors for outgoing trajectories that may go through $\widehat{R}$ (Fig. 1: the trajectory from C). This way, changes propagate through the abstract state space.

We now define the algorithm formally. First, define the set of *neighbors* of an abstract state $\widehat{x} \in \widehat{X}$ as

$$\mathcal{N}(\widehat{x}) := \{\widehat{x}' \in \widehat{X} \mid |ctr(\widehat{x}') - ctr(\widehat{x})| \leq \eta\}. \quad (6)$$

We generalize $\mathcal{N}$ for a set of states $P \subseteq \widehat{X}$ by defining

$$\mathcal{N}(P) := \bigcup_{\widehat{x} \in P} \mathcal{N}(\widehat{x}). \quad (7)$$

Alg. 2 gives the algorithm for local re-computation of transitions due to changes in the disturbance. The algorithm uses a queue $\mathcal{Q}$—the frontier—of state-input pairs $(\widehat{x}, \widehat{u})$ for which transitions need to be recomputed, and uses a set $done \subseteq \widehat{X} \times \widehat{U}$ to keep track of previously queued $(\widehat{x}, \widehat{u})$ pairs. Initially, all pairs $(\widehat{x}, \widehat{u})$ s.t. $(\widehat{x}, \widehat{u}) \in \widehat{R}$ or $\exists \widehat{x}'.\, (\widehat{x}', \widehat{u}) \in \widehat{R} \wedge \widehat{x} \in \mathcal{N}(\widehat{x}')$ are enqueued in $\mathcal{Q}$ and are also stored in $done$ (Line 2-3).

The algorithm iteratively processes state and input pairs from the queue, updates the transition relation, and propagates the effect of changes by adding more pairs to the queue. The while loop (Lines 4-17) dequeues state-input pairs one by one from $\mathcal{Q}$ (Line 5), deletes the associated old transitions (Line 6), computes new transitions (Line 7-8), and extends the frontier by queuing new state-input pairs (Lines 9-16).

The queuing operation keeps track of the state-input pairs for which some trajectories may cross $\widehat{R}$, possibly in transience. This bookkeeping is taken care of by the procedure called COMPUTEGROWTH. If $isOverlap$ is true for some state-input pair $(\widehat{x}, \widehat{u})$, then all the states in $\mathcal{N}(\widehat{x})$ are suspected to have some outgoing transitions entering $\widehat{R}$ as well. So all the states in $\mathcal{N}(\widehat{x})$, which have not been queued before (checked using $done$), are queued to $\mathcal{Q}$. The process continues until $\mathcal{Q}$ is empty.

The following theorem states that Alg. 2 is sound: it computes the same abstract transition system as the standard abstraction procedure working from scratch.

Theorem 2: Let $\Sigma = (X, U, d, f)$ and $\Sigma' = (X, U, d', f)$ be two control systems. Let R and $\widehat{R}$ be given as in (4) and (5) respectively. For parameters τ and η, let $\widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f})$ and $\widehat{\Sigma}' = (\widehat{X}, \widehat{U}, \widehat{f}')$ be the associated finite state abstractions. Then $\widehat{\Sigma}' = \text{LOCALABS}(\Sigma', \widehat{\Sigma}, \widehat{R}, \eta, \tau)$.

Proof: We need to show that any state-input pair $(\widehat{x}, \widehat{u})$ which has some outgoing transition entering $\widehat{R}$ is eventually queued in $\mathcal{Q}$. Assume, towards a contradiction, that there is a state-input pair $(\widehat{x}, \widehat{u})$ which is never queued in $\mathcal{Q}$, and there is a trajectory $\xi_{\widehat{u}}(\cdot)$ and some $t \in [0, \tau]$ s.t. $\xi_{\widehat{u}}(0) \in \widehat{x}$ and $\xi_{\widehat{u}}(t) \in \cup_{\widehat{x} \in \widehat{R}} \widehat{x}$. Let us generalize $\mathcal{N}(\cdot)$ to multi-step neighbors: $\mathcal{N}^1(P) := \mathcal{N}(P)$ and $\mathcal{N}^{i+1}(P) := \mathcal{N}(\mathcal{N}^i(P))$. By our assumption it must be the case that there exist i, j with $i < j$ s.t. there exists $\widehat{y} \in \mathcal{N}^j(\widehat{x})$ with $(\widehat{y}, \widehat{u})$ being queued in $\mathcal{Q}$, and there is no $\widehat{z} \in \mathcal{N}^i(\widehat{x})$ s.t. $(\widehat{z}, \widehat{u})$ is ever queued in $\mathcal{Q}$. This is only possible if there is a discontinuity in $\xi_{\widehat{u}}(\cdot)$, which is a contradiction. ∎

Algorithm 1 Procedure GLOBALABS

Input: $\Sigma = (X, U, f, d), \widehat{U} \subseteq U, \eta, \tau$
Output: $\widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f})$

1: $\widehat{X} \leftarrow$ a cover of X with discretization parameter η
2: $\widehat{f} \leftarrow \emptyset$
3: **for all** $(\widehat{x}, \widehat{u}) \in \widehat{X} \times \widehat{U}$ **do**
4: $\quad (x', r', \cdot) = \text{COMPUTEGROWTH}(ctr(\widehat{x}), \widehat{u}, \eta, \tau, \cdot)$
5: $\quad \widehat{f} \leftarrow \widehat{f} \cup \{(\widehat{x}, \widehat{u}, \widehat{x}') \mid [\![x' - r', x' + r']\!] \cap \widehat{x}' \neq \emptyset\}$
6: **end for**
7: **return** $\widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f})$

Algorithm 2 Procedure LOCALABS

Input: $\Sigma' = (X, U, f, d'), \widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f}), \widehat{R}, \eta, \tau$
Output: $\widehat{\Sigma}' = (\widehat{X}, \widehat{U}, \widehat{f}')$

1: $\widehat{f}' \leftarrow \widehat{f}$
2: $\mathcal{Q} \leftarrow \widehat{R} \cup \{(\widehat{x}, \widehat{u}) \mid \exists \widehat{x}'.\, (\widehat{x}', \widehat{u}) \in \widehat{R} \wedge \widehat{x} \in \mathcal{N}(\widehat{x}')\}$
3: $done \leftarrow \widehat{R} \cup \{(\widehat{x}, \widehat{u}) \mid \exists \widehat{x}'.\, (\widehat{x}', \widehat{u}) \in \widehat{R} \wedge \widehat{x} \in \mathcal{N}(\widehat{x}')\}$
4: **while** $\mathcal{Q} \neq \emptyset$ **do**
5: $\quad (\widehat{x}, \widehat{u}) \leftarrow \mathcal{Q}.dequeue()$
6: $\quad \widehat{f}' \leftarrow \widehat{f}' \setminus \{(\widehat{x}, \widehat{u}, \widehat{x}') \mid (\widehat{x}, \widehat{u}, \widehat{x}') \in \widehat{f}\}$
7: $\quad (x', r', isOverlap) =$
$\qquad\quad \text{COMPUTEGROWTH}(ctr(\widehat{x}), \widehat{u}, \eta, \tau, \widehat{R})$
8: $\quad \widehat{f}' \leftarrow \widehat{f}' \cup \{(\widehat{x}, \widehat{u}, \widehat{x}') \mid [\![x' - r', x' + r']\!] \cap \widehat{x}' \neq \emptyset\}$
9: $\quad$ **if** $isOverlap$ **then**
10: $\quad\quad$ **for all** $\widehat{y} \in \mathcal{N}(\widehat{x})$ **do**
11: $\quad\quad\quad$ **if** $(\widehat{y}, \widehat{u}) \notin done$ **then**
12: $\quad\quad\quad\quad \mathcal{Q}.enqueue(\widehat{y}, \widehat{u})$
13: $\quad\quad\quad\quad done \leftarrow done \cup \{(\widehat{y}, \widehat{u})\}$
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end for**
16: $\quad$ **end if**
17: **end while**
18: **return** $\widehat{\Sigma}' = (\widehat{X}, \widehat{U}, \widehat{f}')$

The following theorem characterizes the time complexity of LOCALABS in terms of the number of state-input pairs processed. The time complexity is bounded by a factor depending on $|\widehat{R}|$ and a factor depending on the speed of the underlying system dynamics (how far a trajectory can go in one sampling period). We introduce the notation $\oplus$ which inflates the boundary of a set $\widehat{R}$, by an extent given by the set $\widehat{S} = [\![-s, s]\!] \in \mathbb{N}^n$:

$$\widehat{R} \oplus \widehat{S} := \{\widehat{x} \in \widehat{X} \mid \exists \widehat{x}' \in \widehat{R}\, . \\ \exists \widehat{y} \in \widehat{S}\, .\, ctr(\widehat{x}) = ctr(\widehat{x}') + \widehat{y} \cdot \eta\}, \quad (8)$$

where the vector product is defined component-wise.

Theorem 3: Let $\Sigma = (X, U, d, f)$ be a control system with $X = \mathbb{R}^n$. For parameters τ and η, define the n-dimensional vector $s := \lceil \sup_{x,u}(|f(x, u) + d(x, u)| \cdot \tau/\eta) \rceil$, and the set $\widehat{S} = [\![-s, s]\!] \in \mathbb{N}^n$, where the divisions and

$\lceil \cdot \rceil$ are defined component-wise. Let the time complexity of COMPUTEGROWTH be some function $h(n)$ of the system dimension n. Then the time complexity of LOCALABS is $\mathcal{O}\left(\min\{|\widehat{X}| \cdot |\widehat{U}|, |\widehat{R} \oplus \widehat{S}|\} \cdot h(n)\right)$.

Proof: The time complexity of LOCALABS depends on the number of state-input pairs queued in $\mathcal{Q}$. This number is proportional to the size of $|\widehat{R}|$ expanded by how far a state can travel in the sampling time τ. Thus, an upper bound on the number of states is $\min\{|\widehat{X}| \cdot |\widehat{U}|, |\widehat{R} \oplus \widehat{S}|\}$. Since each state-input pair is queued only once, and for each such queued pair the procedure COMPUTEGROWTH is run only once, hence the time complexity of LOCALABS is $\mathcal{O}\left(\min\{|\widehat{X}| \cdot |\widehat{U}|, |\widehat{R} \oplus \widehat{S}|\} \cdot h(n)\right)$. ∎

In contrast, the run time of GLOBALABS is $\mathcal{O}\left(|\widehat{X}| \cdot |\widehat{U}| \cdot h(n)\right)$. Thus, when $|\widehat{R} \oplus \widehat{S}| \ll |\widehat{X}| \cdot |\widehat{U}|$, LOCALABS is significantly faster than GLOBALABS, and when $|\widehat{R} \oplus \widehat{S}| \approx |\widehat{X}| \cdot |\widehat{U}|$, LOCALABS and GLOBALABS have comparable performance. (The trend is clearly visible in Fig. 4 where we compare computation times of LOCALABS and GLOBALABS as a function of $|\widehat{R}|/(|\widehat{X}| \cdot |\widehat{U}|)$ for a vehicle example.)

Remark 1 (Selective enqueuing): As apparent from the proof of Theorem 3, a simple over-approximation of the set of state-input pairs to be enqueued in $\mathcal{Q}$ in Alg. 2 could be $\{\widehat{R} \oplus \widehat{S}\} \times \widehat{U}$. This over-approximation might lead to enqueuing spurious states from which no transitions enter $\widehat{R}$ in time τ, and hence they do not require re-computation. Our algorithm LOCALABS dynamically and selectively enqueues state-input pairs only when it's absolutely necessary. The benefit of this selective enqueuing is most prominent if the speed of the system dynamics is not uniform in all directions, in which case the estimate $\{\widehat{R} \oplus \widehat{S}\} \times \widehat{U}$ is overly conservative.

V. A NOTE ON THE PARALLEL COMPUTATION OF ABSTRACTION UPDATES

ABCS methods are ultimately limited by the size of the model's state and input spaces. Unfortunately, the number of hypercubes in the abstraction is exponential in the number of dimensions. In order to handle larger examples, the computation needs to be parallelized. Fortunately, the computation of transitions for the state-input pairs are mutually independent, hence they can be executed in parallel. This is extremely useful for scaling ABCS to very large dimensional systems by maximally utilizing the available computational resources. Our largest example has 12 dimensions and our implementation runs on a 192-cores machine (see Section VII).

Parallelizing GLOBALABS is straightforward. The for-loop (Alg. 1 lines 3-5) is replaced by a parallel for loop.

For the LOCALABS algorithm, the situation is more complicated. The while-loop in Line 4 of Alg. 2 is not as straightforward to parallelize. This is because a state-input pair dequeued from $\mathcal{Q}$ comes from a previous iteration of the same loop. $\mathcal{Q}$ is a shared data structure and, therefore, modifying it requires extra precautions. Our implementation alleviates this by enqueuing and dequeuing batches of state-input pairs but some overhead remains. $\mathcal{Q}$ is still a major point of contention.

Algorithm 3 Parallelized Procedure SEMILOCALABS

Input: $\Sigma' = (X, U, f, d'), \widehat{\Sigma} = (\widehat{X}, \widehat{U}, \widehat{f}), \widehat{R}, \eta, \tau$
Output: $\widehat{\Sigma}' = (\widehat{X}, \widehat{U}, \widehat{f}')$

1: $\widehat{f}' \leftarrow \widehat{f}$
2: **parfor** $(\widehat{x}, \widehat{u}) \in \{\widehat{R} \oplus \widehat{S}\} \times \widehat{U}$ **do**
3: $\quad \widehat{f}' \leftarrow \widehat{f}' \setminus \{(\widehat{x}, \widehat{u}, \widehat{x}') \mid (\widehat{x}, \widehat{u}, \widehat{x}') \in \widehat{f}\}$
4: $\quad (x', r', \cdot) = \text{COMPUTEGROWTH}(ctr(\widehat{x}), \widehat{u}, \eta, \tau, \cdot)$
5: $\quad \widehat{f}' \leftarrow \widehat{f}' \cup \{(\widehat{x}, \widehat{u}, \widehat{x}') \mid [\![x' - r', x' + r']\!] \cap \widehat{x}' \neq \emptyset\}$
6: **end parfor**
7: **return** $\widehat{\Sigma}' = (\widehat{X}, \widehat{U}, \widehat{f}')$

To get around this problem we adopt a more pragmatic approach. We can recompute more state-input pairs as long as we save more time because of less synchronization. We simply use the idea as briefed in Rem. 1 to restrict the GLOBALABS procedure to a smaller region. The final algorithm SEMILOCALABS is shown in Alg. 3.

Although this is theoretically sub-optimal, it is simpler to implement and can be faster than using a shared queue. More precisely, when the speed of the dynamics is similar across the whole space, $\{\widehat{R} \oplus \widehat{S}\} \times \widehat{U}$ is a rather precise over-approximation of what needs to be recomputed. On the other hand, as $\widehat{S}$ is computed over the whole state space, if $\widehat{R}$ is a region where the dynamics are slower than in other parts of the state space SEMILOCALABS will not be as good. We compare the different algorithms in Table I. It can be seen that for the vehicle model, SEMILOCALABS explores the whole state space due to a large enough value of τ, and the run-time is also much worse than LOCALABS. On the other hand for the quadrotor model, even though SEMILOCALABS explores more states, it performs better than LOCALABS.

Our parallelized implementation relies on OpenMP and runs on a single, albeit large, machine. We believe it can be parallelized further. Recently, Khaled et al. presented a parallel implementation of ABCS in their tool pFaces [13]. Their parallelization goes further, distributes the computation across multiple machines, and uses GPUs.

VI. A NOTE ON INCREMENTAL ABCS

Within the ABCS work-flow, the computed abstract transition system is the basis for discrete controller synthesis. The resulting controller is further refined into a continuous one to actuate the underlying dynamical system. This implies, that a local update of the abstract transition system induced by our new technique might also change the corresponding controller synthesis problem leading to a different control implementation. E.g., for the motion control of a unicycle robot depicted in Fig. 1, the disturbance update in region C2 resulted in a new control policy which made the robot move through region C1 to reach the target (B), while it was aiming to pass C2 before the abstraction update.

Similar to locally updating the abstraction, control policies computed via reactive synthesis can also be updated dynamically. This has been shown by Chatterjee and Henzinger [7] in the context of finite state games to implement Büchi

objectives. While we expect that the algorithm in [7] can be readily transferred to the setting of ABCS, our current implementation re-computes control strategies from scratch. In our examples, this has little effect on the overall performance due to relatively fast synthesis step. For instance, the synthesis takes only $0.3s$ for the 3-dimensional vehicle model (see Section I), $17s$ for the 12-dimensional quadrotor model, and $2s$ for the 5-dimensional vehicle (see Section VII) compared to the much higher respective abstraction time.

VII. Experimental Results

We implemented GlobalAbs, LocalAbs, and SemiLocalAbs as an extension of SCOTSv0.2 [20] available on github[3]. We evaluated our algorithms on two benchmark examples with different dynamics to show the applicability of our method. Beyond the incremental algorithms and the parallelization, we also modified the procedure computing the transitions to account for state-input dependent disturbances [3]. The code base is written in C++ and parallelized using OpenMP. All the experiments were performed on an Intel(R) Xeon(R) Processors E7-8850 v2 (2.30GHz) with 192 cores and 2048 GB memory. A performance comparison of different algorithms is given in Table I. Further, Fig. 4 shows how algorithms perform w.r.t. a varying size of the region where the disturbance change.

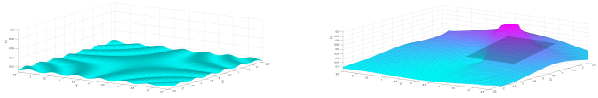
A. Quadrotor Dynamics

We consider the dynamics of a quadrotor $\dot{x} \in f(x,u) + [\![-d(x_1,x_2), d(x_1,x_2)]\!]$, where the state x and the disturbance function d are 12-dimensional vectors, u is 4-dimensional, and d is only assumed to depend on x_1 and x_2. The nominal dynamics are given by

$$f(x,u) = \begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ \frac{\cos x_6}{\sin x_5} x_{11} + \frac{\cos x_6}{\cos x_5} x_{12} \\ x_{11}\cos x_6 - x_{12}\sin x_6 \\ x_{10} + x_{11}\sin x_6 \tan x_5 + x_{12}\cos x_6 \tan x_5 \\ -\frac{1}{m}(\sin x_6 \sin x_4 + \cos x_6 \cos x_4 \sin x_5)\Omega_1 \\ -\frac{1}{m}(\cos x_4 \sin x_6 - \cos x_6 \sin x_4 \sin x_5)\Omega_1 \\ g - \frac{1}{m}(\cos x_6 \cos x_5)\Omega_1 \\ \frac{I_y - I_z}{I_x} x_{11} x_{12} + \frac{1}{I_x}\Omega_2 \\ \frac{I_z - I_x}{I_y} x_{10} x_{12} + \frac{1}{I_y}\Omega_3 \\ \frac{I_x - I_y}{I_z} x_{10} x_{11} + \frac{1}{I_z}\Omega_4 \end{bmatrix},$$

and $\Omega_1 = bl(u_1^2 + u_2^2 + u_3^2 + u_4^2)$, $\Omega_2 = bl(u_3^2 - u_1^2)$, $\Omega_3 = bl(u_4^2 - u_2^2)$, $\Omega_4 = d(u_2^2 + u_4^2 - u_1^2 - u_3^2)$ for certain constant parameters $m, g, b, l, d, I_x, I_y, I_z$. The bounds on the state variables are given by $x_1 \in [-2.5, 2.5]$, $x_2 \in [-2.5, 2.5]$, $x_3 \in [-2.5, 2.5]$, $x_4 \in [\frac{-30\pi}{180}, \frac{30\pi}{180}]$, $x_5 \in [\frac{-30\pi}{180}, \frac{30\pi}{180}]$, $x_6 \in [\frac{-30\pi}{180}, \frac{30\pi}{180}]$, $x_7 \in [0, 0.9]$, $x_8 \in [0, 0.9]$, $x_9 \in [0, 0.9]$, $x_{10} \in [-0.05, 0.05]$, $x_{11} \in [-0.05, 0.05]$, and $x_{12} \in [-0.05, 0.05]$. The bounds on the control inputs are given by $u_1 \in [100, 2500]$, $u_2 \in [100, 2500]$, $u_3 \in [100, 2500]$, and $u_4 \in [100, 2500]$. A more detailed explanation of the system model and the physical meaning of the variables can be found in [6].

[3] https://github.com/YunjunBai/online_controller



(a) Initial disturbance model (b) New disturbance model

Fig. 3. Disturbance model in dimension 11 for the quadrotor example. X- and Y-axis represent state variable x_1 and x_2. Z-axis represents d_{11}. (a) shows the disturbance model used to compute the initial abstraction. (b) shows the updated disturbance model. The shaded region in (b) is $\widehat{R}$.

Initially, the disturbances $d_i(\cdot,\cdot)$ for $i \notin \{10, 11\}$ are assumed to be uniformly zero, and $d_{10}(x_1,x_2) = d_{11}(x_1,x_2) = 0.123$ for all x_1, x_2. Later, the disturbance locally changes to d' in region $\widehat{R} = [0.5, 2] \times [-2, 0]$, s.t. for $(x_1,x_2) \in \widehat{R}$, $d'_{10}(x_1,x_2) = 0.123$ (same as d), $d'_{11}(x_1,x_2) = d'_{12}(x_1,x_2) = 0.523$ (updated), and $d'_i(x_1,x_2) = 0$ for all $i \notin \{10, 11, 12\}$ (same as d). For $(x_1,x_2) \notin \widehat{R}$, we assume that $d'(x_1,x_2) = d(x_1,x_2)$. For technical reasons, the disturbance function needs to be continuous in the state space. We assume that d' is actually modeled by a continuous function (e.g. using a rapidly changing sigmoid function) in the boundary of $\widehat{R}$.

Note that $\widehat{R}$ only covers approximately 12.5% of the entire state-space, and so a complete re-computation of abstraction when d is changed to d' is wasteful. This fact can be seen from Table I by observing that the runtime for LocalAbs is much smaller than that for GlobalAbs .

The parameters that we use for the abstraction are given by $\omega = (400, 400, 400, 400)$, $\tau = 1 \times 10^{-3}s$, and $\eta = (1.5, 1.5, 1.5, \frac{20\pi}{180}, \frac{20\pi}{180}, \frac{20\pi}{180}, 0.3, 0.3, 0.3, 0.05, 0.05, 0.05)$.

B. Vehicle Dynamics

We consider a 5-dimensional model $\dot{x} \in f(x,u) + [\![-d(x_1,x_2), d(x_1,x_2)]\!]$ of a vehicle, where the disturbance d is a function of x_1, x_2 and the nominal model is given by

$$f(x,u) = \begin{bmatrix} \alpha_1 x_4 + \alpha_2 x_5 \\ \alpha_3 x_4 + \alpha_4 x_5 \\ \frac{r}{2b}(x_5 - x_4) \\ -d_1 x_4 + b_1 u_1 + b_0 u_2 \\ -d_2 x_5 + b_0 u_1 + b_1 u_2, \end{bmatrix}$$

with $\alpha_1 = 0.5 r x_4 (\cos x_3 + a \sin x_3)$, $\alpha_2 = 0.5 r x_4 (\cos x_3 - a \sin x_3)$, $\alpha_3 = 0.5 r x_4 (\sin x_3 - a \cos x_3)$, $\alpha_4 = 0.5 r x_4 (\sin x_3 + a \cos x_3)$. The bounds on the state variables are given by $x_1, x_2, x_4, x_5 \in [0, 10]$, $x_3 \in [-3.5, 3.5]$, and on the input variables by $u_1, u_2 \in [0, 6]$.

Initially, the disturbances are given by $d_4(x_1,x_2) = d_5(x_1,x_2) = 0.01$ and $d_i(x_1,x_2) = 0$ for $i \in \{1, 2, 3\}$. Later, the disturbance locally changes to d' in region $\widehat{R} = [4, 7] \times [2, 5]$, s.t. for $(x_1,x_2) \in \widehat{R}$, $d'_4(x_1,x_2) = d'_5(x_1,x_2) = 0.1$ (updated), and $d'_i(x_1,x_2) = 0$ for all $i \in \{1, 2, 3\}$ (same as d). For $(x_1,x_2) \notin \widehat{R}$, we assume that $d'(x_1,x_2) = d(x_1,x_2)$. As before, we assume that d' is actually modeled by a continuous function (e.g. using a rapidly changing sigmoid function) in the boundary of $\widehat{R}$.

TABLE I

EXPERIMENTAL COMPARISON OF GLOBALABS, LOCALABS AND SEMILOCALABS. IT SHOWS THE NUMBER OF STATES AND INPUTS, THE SIZE OF THE RE-COMPUTED REGION IN PERCENT AND THE RUNTIME OF GLOBALABS (LEFT). THE PERFORMANCE OF LOCALABS (MIDDLE) AND SEMILOCALABS (RIGHT) IS SHOWN IN TERMS OF RUNTIME, RE-COMPUTED PORTION OF THE STATE SPACE, AND SPEEDUP W.R.T. GLOBALABS.

System	$\lvert\widehat{X}\rvert$	$\lvert\widehat{U}\rvert$	$\lvert\widehat{R}\rvert/(\lvert\widehat{X}\rvert\cdot\lvert\widehat{U}\rvert)$	GLOBALABS	LOCALABS			SEMILOCALABS		
				time (s)	time (s)	re-computation	relative	time (s)	re-computation	relative
Quadrotor	1 259 712	625	12.5%	815.34	234.54	18.75%	3.5×	154.67	33.33%	5.3×
Vehicle	195 657	49	9%	34.27	8.12	14.4%	4.2×	35.46	100%	1×

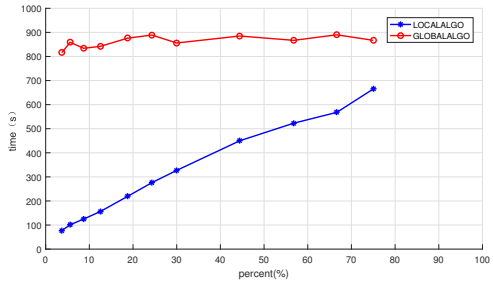
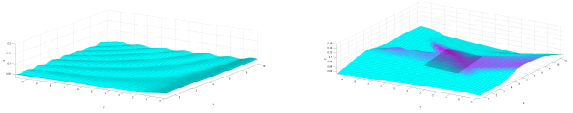


Fig. 4. Computation time of LOCALABS and GLOBALABS for different sizes of $\widehat{R}$ in percentage of the state space in the quadrotor example. LOCALABS is more efficient the smaller $\widehat{R}$.



(a) Initial disturbance model (b) New disturbance model

Fig. 5. Disturbance model in dimension 4 for the vehicle example. X- and Y-axis represent state variables x_1 and x_2. Z-axis represents d_4. (a) shows the disturbance model used to compute initial abstraction. (b) shows the updated disturbance model. The shaded region in (b) is $\widehat{R}$.

The parameters used for the abstraction are: $\eta = (1, 1, 1, 0.5, 1)$, $\omega = (1, 1)$, and $\tau = 0.3s$.

It can be seen in Table I (second line, middle) that when the disturbance changes in 9% of the state space, then LOCALABS performs 4 times faster than GLOBALABS.

VIII. CONCLUSION

We have presented a technique for adapting system abstractions to local changes in the external perturbations or modeling uncertainties in the context of abstraction-based controller synthesis. When the perturbation changes in some parts of the state space, our adaptation procedure performs a local re-computation of transitions within the existing abstraction. This results in a substantial speed-up against complete re-computation especially when the region with an updated disturbance is small compared to the whole state space. We also presented a non-trivial parallelized version of our adaptation procedure. We showed the computational benefit of our algorithm using two benchmark examples.

REFERENCES

[1] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR'97*, LNCS 1466, pages 163–178. Springer, 1998.

[2] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin. Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226, 2013.

[3] Y. Bai, R. Majumdar, K. Mallik, A.-K. Schmuck, and D. Zufferey. Accurate abstractions for controller synthesis with non-uniform disturbances. `https://people.mpi-sws.org/~kmallik/papers/NonUniformGrowthBound.pdf`, 2019.

[4] C. Belta, B. Yordanov, and E. A. Gol. *Formal Methods for Discrete-Time Dynamical Systems*. Springer, 2017.

[5] F. Berkenkamp, R. Moriconi, A. P. Schoellig, and A. Krause. Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes. In *CDC*, pages 4661–4666. IEEE, 2016.

[6] T. Bresciani. Modelling, identification and control of a quadrotor helicopter. *MSc Theses*, 2008.

[7] K. Chatterjee and M. Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *JACM*, 61(3):15, 2014.

[8] J. Daum, Á. Torralba, J. Hoffmann, P. Haslum, and I. Weber. Practical undoability checking via contingent planning. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

[9] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *IEEE TAC*, 2018.

[10] A. Girard. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*, 48(5):947–953, 2012.

[11] J. Hoffmann. Where "ignoring delete lists" works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

[12] K. Hsu, R. Majumdar, K. Mallik, and A. Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *HSCC'18*, pages 120–129. ACM, 2018.

[13] M. Khaled and M. Zamani. pfaces: An acceleration ecosystem for symbolic control. In *HSCC'19*, 2019.

[14] J. Liu and N. Ozay. Finite abstractions with robustness margins for temporal logic-based control synthesis. *Nonlinear Analysis: Hybrid Systems*, 22:1–15, 2016.

[15] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS'95*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.

[16] M. Mazo Jr., A. Davitian, and P. Tabuada. PESSOA: A tool for embedded controller synthesis. In *CAV 2010*, LNCS 6174, pages 566–569. Springer, 2010.

[17] P. Nilsson, N. Ozay, and J. Liu. Augmented finite transition systems as abstractions for control synthesis. *Discrete Event Dynamic Systems*, 27(2):301–340, 2017.

[18] J. F. Quindlen, U. Topcu, G. Chowdhary, and J. P. How. Region-of-convergence estimation for learning-based adaptive controllers. In *ACC*, pages 2500–2505. IEEE, 2016.

[19] G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *TAC*, 62(4):1781–1796, 2017.

[20] M. Rungger and M. Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *HSCC'16*, pages 99–104. ACM, 2016.

[21] S. Sadraddini and C. Belta. Formal methods for adaptive control of dynamical systems. In *CDC*, pages 1782–1787. IEEE, 2017.

[22] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer, 2009.