Automatic Synthesis of Robust Numerical Controllers

Giuseppe Della Penna, Daniele Magazzeni, Alberto Tofani
Department of Computer Science
University of L'Aquila, Italy
{dellapenna,magazzeni,tofani}@di.univaq.it

Benedetto Intrigila
Department of Mathematics
University of Roma "Tor Vergata", Italy
intrigil@mat.uniroma2.it

Igor Melatti, Enrico Tronci
Department of Computer Science
University of Roma "La Sapienza", Italy
{melatti,tronci}@di.uniroma1.it

Abstract

A major problem of numerical controllers is their robustness*, i.e. the state read from the plant may not be in the controller table, although it may be* close *to some states in the table. For continuous systems, this problem is typically handled by interpolation techniques. Unfortunately, when the plant contains both continuous and discrete variables, the interpolation approach does not work well.*

To cope with this kind of systems, we propose a general methodology that exploits explicit model checking in an innovative way to automatically synthesize a (time-) optimal numerical controller from a plant specification and apply an optimized strengthening algorithm only on the most significant states, in order to reach an acceptable robustness degree.

We implemented all the algorithms within our CGMurφ tool, an extension of the well-known CMurφ verifier, and tested the effectiveness of our approach by applying it to the well-known truck and trailer obstacles avoidance *problem.*

1 Introduction

A control system (or, shortly, *controller*) is a hardware/-software component that controls the behavior of a larger system, called *plant*. In a closed loop configuration, the controller reads the plant state (looking at its *state variables*) and adjusts its *control variables* in order to keep it in a particular state, called *setpoint*, which represents its *normal* or *correct* behavior.

A *numerical controller* is, in essence, a table, indexed by the plant states, whose entries are commands for the plant. These commands are used to set the control variables in order to reach the setpoint from the corresponding states. Namely, when the controller reads a state from the plant, it looks up the action described in the associated table entry and sends it to the plant.

A major problem of numerical controllers is their *robustness*. A controller is robust if it is able to handle all the possible states of the plant. Indeed, due to approximation of continuous variables, plants unavoidably present states that are not known to the controller, although they may be very *close* to some states in the table. For continuous systems, this problem is typically handled by interpolation techniques (e.g. see [11]). Unfortunately, when the plant contains both continuous and discrete variables (i.e., it is a *hybrid system*), the interpolation approach does not work well. Therefore, in this case other approaches should be applied (e.g. see ([13, 10])).

As an example, consider the well known *truck and trailer with obstacles avoidance* problem. In this problem, the goal of the controller is to back a truck with a trailer up to a parking place starting from any initial position in the parking lot and in the presence of obstacles to avoid. Note that we assume a *realistic* formulation of the problem, where the topology of obstacles is given *in a tabular way* (e.g., obtained through an automatic obstacles detection system). This problem is known to be very hard because of the following reasons:

- first of all, the dynamics of the truck-trailer is very complex and, even if the truck-trailer dynamics satisfies the Lipschitz condition ([1, 7]), the presence of obstacles makes the problem *non-Lipschitz* (in particular the transition function is not even *continuous*); thus, it cannot be solved using analytical methods;
- moreover, this problem is hard to solve using a dynamic programming approach ([3, 2]), since a backward decomposition of the cost function (e.g., the

length of the path) is hard to perform, due to the complexity of the system dynamics and to the presence of obstacles;

- finally, local heuristics (e.g. fuzzy rules) perform poorly in this context, since the presence of obstacles may make a good *local* maneuver not suitable for the final goal.

Therefore, in this case, even the synthesis of a good numerical controller (not to say a robust one) seems to be computationally difficult.

To cope with this kind of systems, in [8], we proposed a general methodology that exploits explicit model checking in an innovative way to automatically synthesize a (time-) optimal numerical controller from a plant specification. Controllers generated through our technique can handle a wide variety of plant states, and perform optimally on the states stored in the table. This means that, when the plant is in a known state, the controller is always able to select the action that will drive it to the setpoint in the least number of steps.

Unfortunately, the approach was not able to solve the controller robustness problem in a really satisfactory way. To this aim, in this paper we present a general methodology that can be applied to any numerical controller to automatically achieve a satisfying robustness degree.

To motivate our approach, consider again the *truck and trailer with obstacles avoidance* problem. A computational analysis of the controller described above, shows that there exist some particular states which are in the controller, but when slightly *perturbed*, i.e. when their position is affected by some *disturbance*, give rise to non-stabilizing trajectories. Indeed, such states are those lying *very near to obstacles*, so that only *very small disturbances* in their actual position are manageable. This holds also for states which are *very near to the so called "jackknife" positions* [8]. From numerical experiments, it results that all these states can be stabilized only in case we allow errors in their position not greater than $0.1m$ for a truck $6m$ long. Since this precision is *not realistic*, it seems more natural to cut off the most *unsafe* states - so to obtain a good robustness for the resulting controller.

Our idea is to use explicit model checking techniques to perform a *probabilistic* analysis of the plant state space in order to detect the *unsafe states*, from which the controller has - in case of disturbances - a very low probability of reaching the setpoint. Then, we exclude such *unsafe* states from our controller and apply a *strengthening algorithm* only to the *safe* (i.e. *not unsafe*) states. This algorithm extends the initial controller to cope with disturbances. After this process, we have that the controller is able, in most cases and even in presence of disturbances, to drive the plant from any *safe* state to a *safe* state stored in the table of the optimal controller and, from there, reach the setpoint.

We implemented this new robustness algorithm in our CGMurφ tool ([5, 8]), obtaining a three-step automatic process for the synthesis of numerical controllers. It is important to notice that while the first phase is computationally very expensive, the two other phases are highly parallelizable. The resulting controllers are time optimal for the states stored in the table and are quite robust w.r.t. the other plant states, with the exception of a small set of *unsafe states*, which may be controllable only in the presence of very small disturbances.

This new approach dramatically improved our results in the *truck and trailer with obstacles avoidance* case study: from a 74-88% of robustness w.r.t. small disturbances [8] to a 91-95% of robustness, even in the presence of big disturbances.

The paper is organized as follows. In Sect. 2 we describe our methodology and give a formal description of our probabilistic approach. In Sect. 3 we present the new version of CGMurφ tool that includes the probabilistic strengthening technique. In Sect. 4 we describe our case study and give experimental results showing the improvement obtained w.r.t the previous approach. Sect. 5 concludes the paper.

2 Optimal and Robust Controller Generation through Model Checking

We suppose to have a plant $\mathcal{P}$ with dynamics $F(x, q, u_c, u_d)$, where $x \in \mathbf{R^n}$ is the vector of the continuous components of the state, $q \in \mathbf{N^k}$ is the vector of the discrete components of the state, $u_c \in \mathbf{R^m}$ is the vector of the continuous components of the control, and $u_d \in \mathbf{N^t}$ is the vector of the discrete components of the control.

The problem of the controllability of $\mathcal{P}$ to a specified state (x_0, q_0), called the *setpoint*, is considered (i) with respect to the continuous variables, in a given bounded region H of $\mathbf{R^n}$, containing a neighborhood of x_0, and (ii) with the discrete variables ranging over a finite subset Q of $\mathbf{N^k}$.

Having continuous and discrete components $\mathcal{P}$ is a *hybrid system*. We always suppose that for fixed values of q and u_d, $F(x, q, u_c, u_d)$ is a continuous function of (x, u_c). Moreover, we assume that only a bounded set $||u_c|| \leq \gamma$ of continuous controls and, respectively, a finite set U_d of discrete controls need to be considered.

After a suitable discretization of the continuous components of both the state and the control, we can assume that we have only a *finite* number of states and, in every state, only a *finite* number of allowed control actions.

Our objective is to build an optimal controller for $\mathcal{P}$, i.e. a controller that is able to drive $\mathcal{P}$ to the setpoint starting from any initial state. Moreover, the robustness constraint

makes us consider also states that may be reached in the presence of errors and disturbances in the state and control variables. Such disturbances may derive, e.g., from sensor noise or real values approximation. Finally, the optimality constraint requires the controller to reach the setpoint within the smallest possible number of steps (*time optimality*).

Thus, our controller has to decide, for every considered plant state, which is the best action (w.r.t. the number of steps) to reach the setpoint. The optimality of the action chosen implies the optimality of the generated controller.

In order to build such a controller, we consider the *transition graph* $\mathcal{G}$ of $\mathcal{P}$, where the nodes are the reachable states and a transition between two nodes models an allowed action between the corresponding states. $\mathcal{G}$ has a given set $\{s_0, s_1, \ldots, s_l\}$ of *initial states* (that is, the states arising from the discretization of the region H). Moreover, $\mathcal{G}$ has a set of *goal states* (that is, the (discretized) states sufficiently near to the setpoint).

In this setting, the problem of designing the optimal controller reduces to finding the minimum path in $\mathcal{G}$ between each initial state and the nearest goal state.

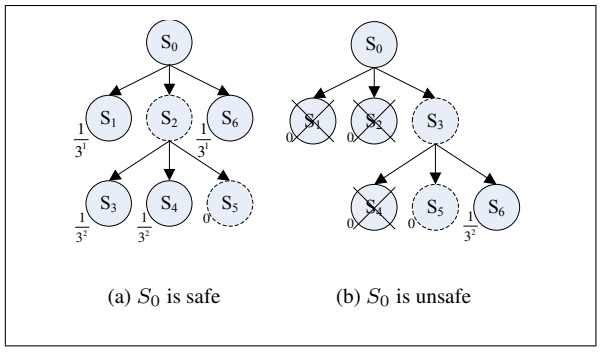
Unfortunately, the transition graph for complex, real-world systems could be often huge, due to the well-known *state explosion* problem. Thus it is likely that $\mathcal{G}$ would not fit into the available RAM memory, and then the minimum path finding process could be highly time consuming. However, model checking techniques [4] developed in the last decades have shown to be able to deal with huge state spaces. Thus, we used model checking algorithms, reshaping them in order to automatically synthesize optimal controllers, as presented in [8], and we implemented this methodology in the CGMurφ tool [5].

More in detail, we have three phases, that we describe in the following.

2.1 Optimal Controller Synthesis

In the first phase, an explicit model checking algorithm is used, which performs $l + 1$ extended depth first visits of all the reachable states of $\mathcal{P}$ starting from each initial state in $\{s_0, s_1, \ldots, s_l\}$. This phase is described in details in [8], here we only sketch the resulting algorithm.

The depth first visit algorithm and data structures are enriched in order to generate the controller $\mathcal{C}$. As usual, a *hash table* is used in order to store already visited states. A *stack* holds, together with the states on the graph path being explored, also the next action to be considered from each state. During the i-th visit, when a goal state g is reached, then the states in the current path from s_i to g are marked, and we remember the actions taken on the path as well as the path length. If more than one goal path starting from the same state is found, we pick the shortest one. Thus, when all visits end, we can insert in $\mathcal{C}$ the shortest path from each initial state to a goal state. The resulting controller is therefore optimal: in fact, it contains a set of *optimal plans* that can be used to drive $\mathcal{P}$ to the goal from any *controllable* state (i.e. a reachable state that is connected to a goal state).



(a) S_0 is safe (b) S_0 is unsafe

Figure 1. Examples of Probability computation.

However, $\mathcal{C}$ does not take into account any state *outside* the optimal plans and therefore it is not robust.

2.2 Probabilistic Analysis of the Controller States

In this second phase we extract a particular set of *significant* plant states from the optimal controller.

To this aim we first calculate, for each controlled state s, the probability $p_c(s)$ that from s, after any sequence of allowed actions, we are still in a state of the controller. This gives us a measure of how much the states deriving from s can be handled by our controller. Since we cannot take into account any possible such sequence, we *approximate* $p_c(s)$ by considering only sequences σ of a given length k_c, thus - starting from s - each time we apply all n possible actions of the controller. In this way we construct a tree $\mathcal{T}$ of sequences.

For a given sequence σ in $\mathcal{T}$, we define $|\sigma|$ as follows: let i, with $1 \leq i \leq k_c$, be the minimum value (if it exists) such that the action $\sigma(i)$ leads to a state in the controller, and we put $|\sigma| = i$; we let $|\sigma|$ undefined otherwise.

Let $\mathcal{T}^*$ be the set of all sequences σ such that $|\sigma|$ is defined. Now we compute

$$p_c(s) = \begin{cases} \sum_{\sigma \in \mathcal{T}^*} \frac{1}{n^{|\sigma|}} & \text{if } \mathcal{T}^* \text{ is not empty;} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Note that the choice of k_c depends on the required degree of accuracy (e.g., we choose $k_c = 4$ for our case study, see section 4). In the following, we suppose k_c to be fixed.

We say that a state s is *unsafe* if $p_c(s) < M_c$, where the *safety threshold* M_c is a small value, typically below 10%. Otherwise, s is *safe*. *Safe* states represent the normal states

of the plant, whereas *unsafe* states correspond to extreme states of the plant, that are practically uncontrollable in case of disturbances.

Therefore, our idea is to identify the set of *safe* states $S = \{s \mid p_c(s) \geq M_c\}$ and concentrate the strengthening process on them.

To compute the $p_c(s)$ of each state s in the controller, we use a breadth first visit from s and we incrementally calculate the probability $p_c(s)$. The exploration of each path is stopped after k_c levels or when a state of the controller is reached. Indeed, from Equation (1) we know that the remaining paths do not contribute to the increment of $p_c(s)$. Moreover, a particular class of states to be considered is the one of *error states* which may or may not exist according to the system under consideration. If an error state is *unrecoverable*, that is we cannot escape from it, we have also to stop the visit at any such state.

An example of this computation is shown in Figure 1 where solid circles are the states in $\mathcal{C}$, dashed circles are the states not in $\mathcal{C}$, barred circles are the error states and for each leaf-node the contribution to $p_c(s)$ is indicated. Supposing that $M_c = 20\%$, in case (a) we have that s_0 is *safe* since $p_c(s_0) = \frac{8}{9} \geq M_c$, whereas in case (b) s_0 is *unsafe* since $p_c(s_0) = \frac{1}{9} < M_c$. Note that, in the case (b), s_0 indeed corresponds to a position of the truck close to an obstacle, thus most of the actions available lead the truck to hit an obstacle; as expected, this result in an error state (in our example, there are three error states s_1, s_2 and s_4, corresponding to different ways to hit the obstacle). Our algorithm takes advantage of such a situation by not expanding the error states, thus $p_c(s_0)$ is incremented only when s_6 (i.e. the only state in the controller in this example) is reached.

Once we have built the set S, we can apply the third phase only on the *safe* states.

2.3 Controller Strengthening

The last phase of our approach performs a *strengthening* of the controller $\mathcal{C}$ generated by the first phase.

As described in the previous Subsection 2.2, in this phase we consider only the set of *safe* states S.

To ensure the robustness of the controller, we explore a larger number of states obtained by randomly perturbing the states in S. That is, for each state $s \in S$ we apply a set of small random changes and obtain a set of new states which, generally speaking, are not in the controller.

Then, from each new state s', we start a breadth first visit of the state space of $\mathcal{P}$ stopping it after a given number k_s of levels or if we find a state s'' such that $p_c(s'') \geq M_l$, for a suitably chosen constant M_l.

The meaning of this procedure is the following: from s' we look for a state which is in the controller and, moreover, is sufficiently *safe*. So in a sense, we use the safe states of the controller as *an extended setpoint*.

Formally, let $N(s')$ be the set of visited states during this visit and let t be a state in $S \cap N(s')$ such that $\forall t' \in S \cap N(s') : p_c(t) \geq p_c(t')$. If $p_c(t) > M_h$, the path from s' to t is stored in $\mathcal{C}$, otherwise we declare s *unsafe* (see Section 2.2). Here, the meaning of constants k_s, M_l and M_h, where it is required that $M_h < M_l$, is: the length of the breadth first visit, the *minimum probability value* that we accept to consider a state *near* to the controller and, respectively, the *maximum probability value* that we consider *too low* to consider a state as *safe*. Note that the choice of the constants k_s, M_l and M_h depends on the required controller accuracy.

After some iterations of this process, we have that $\mathcal{C}$ is able to drive $\mathcal{P}$ from any reasonable system state to the *best* near state of the optimal controller and, from there, reach a goal. That is, $\mathcal{C}$ has been augmented with new $(state, action)$ pairs in order to deal with a larger number of possible plant states. This makes it more robust without too much affecting its optimality.

3 The Controller Generator Murφ Tool

The CGMurφ tool [5] is an extended version of the CMurφ ([6, 9]) model checker. It is based on an explicit enumeration of the state space, and it was originally developed to verify protocol-like systems.

In order to generate controllers for complex systems we added to CGMurφ some important features, i.e. finite precision real numbers handling (to model systems with continuous variables) and external linking to C/C++ functions (to easily model the most complex aspects of the plant, or even to interface the model with a complete plant simulator).

In the new version of CGMurφ (available on [5]), we implemented the probabilistic analysis of the controller states as well as the new strengthening phase, while the first phase (the optimal controller synthesis) is the same as in previous version (details can be found in [8]). Therefore, in the following we give details only for the new algorithms, namely the probabilistic analysis and the strengthening algorithms.

Both algorithms work on a main data structure called `CTRL`, that holds the controller table together with a set of variables needed to process it. In particular, `CTRL` contains, for each reachable system state s that leads (in one or more steps) to a goal, a pair (r, c) indicating that the shortest path leading from s to a goal state has c steps, where the first step is the action given by rule r. Moreover, to compute the probability p_c (see Section 2.2), we added to each state in `CTRL` a `prob` variable, initially set to 0.

The algorithm for probabilistic analysis of the controller states performs a breadth first visit starting from each state in the controller table. In particular, in each step we extract a state s from the breadth first visit queue and generate a

set of new states by applying *a specified set* of the allowed actions on it. The newly generated states that are *not* in the controller table are enqueued, whereas those already controlled are used to update the $p_c(s)$.

The algorithm for the controller strengthening performs a number of breadth first visits, each starting from a state obtained by disturbing each controllable state in the controller table. The final outcome is that we add paths to the controller, in order to be able to control also these disturbed states.

Note that both the probabilistic analysis and the strengthening algorithms are *highly parallelizable*. Indeed, the controller states can be partitioned in several subsets and processed simultaneously by different processes (possibly on different machines).

4 Case Study

To measure the effectiveness of our new strengthening approach, we applied it to the *truck and trailer with obstacles avoidance* problem already addressed in [8]. In this way, we will also show the improvements w.r.t. the previous strengthening approach used in [8].

4.1 Problem Definition

We want to synthesize a controller that is able to back a truck with a trailer up to a specified parking place starting from any initial position in the parking lot. Moreover, the parking lot contains some obstacles, which have to be avoided by the truck while maneuvering to reach the parking place. The obstacles position and geometry are given in a tabular way, i.e. each obstacle is a composition of bidimensional figures defined through the position of their vertexes relative to the parking lot origin. This is a reasonable representation that could be automatically generated, e.g. by analyzing an image of the parking lot. We also disallow *corrective maneuvers*, that is the truck cannot move forward to *backtrack* from an erroneous move.

In this setting, finding a suitable maneuver to reach the goal from any starting position is a hard task. On the other hand, as pointed out in the Introduction, finding an *optimal* maneuver is a *very* complex problem, that cannot be modeled and resolved using common mathematical or programming strategies.

Moreover, note that the states of this system contain both continuous variables (e.g., the truck position) and discrete ones (e.g., the boolean variable that indicates if the truck has hit an obstacle). In such a *hybrid system* interpolation techniques cannot be use to obtain a robust controller.

In the following sections, after giving more details about the truck and trailer model, we show the results obtained by applying the methodology described in Section 2 to synthesize an optimal and robust numerical controller for this problem.

4.2 Model Description

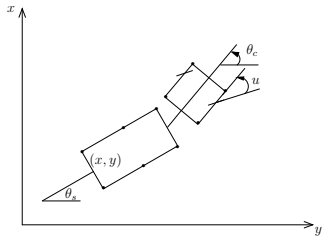


Figure 2. Truck and Trailer System Description

Our model of the truck and trailer is based on the set of equations presented in [12]. Moreover, in our setting the *parking region* is an open bounded region of $\mathbf{R^2}$, delimited by a set of obstacles. We call Γ the parking region. The system has four state variables relative to its position in Γ: the coordinates of the center rear of the trailer ($x, y \in [0, 50]$), the angle of the trailer w.r.t. the x-axis ($\theta_S \in [-90°, 270°]$) and the angle of the cab w.r.t the x-axis ($\theta_C \in [-90°, 270°]$). Moreover the system has a boolean *status variable* q, which has two possible values: `operation`, when the truck lies in Γ, and, respectively, `error`, otherwise.

We assume that the truck moves backward with constant speed of $2m/s$, so the only control variable is the steering angle $u \in [-70°, 70°]$. Figure 2 shows a schematic view of the truck and trailer system with its state and control variables. We single out ten points on the truck and trailer border (displayed in the Figure 2 by bold points) as *representative* of the truck and trailer position.

If the values of the state variables at time t are $q[t] =$ `operation`, $x[t]$, $y[t]$, $\theta_S[t]$ and $\theta_C[t]$, and the steering angle is u, then the new values of state variables at time $t+1$ are determined by following equations:

$$\begin{aligned} x[t+1] &= x[t] - B * cos(\theta_S[t]) \\ y[t+1] &= y[t] - B * sin(\theta_S[t]) \\ \theta_S[t+1] &= \theta_S[t] - arcsin\left(\frac{A*sin(\theta_C[t]-\theta_S[t])}{L_S}\right) \\ \theta_C[t+1] &= \theta_C[t] + arcsin\left(\frac{r*sin(u)}{L_S+L_C}\right) \\ q[t+1] &= q[t] \end{aligned} \tag{2}$$

where $A = r * cos(u)$, $B = A * cos(\theta_C[t] - \theta_S[t])$, $r = 1$ is the truck movement length per time step, $L_S = 4$ and $L_C = 2$ are the length of the trailer and cab, respectively (all

the measures are in meters), *provided that*: $(x[t+1], y[t+1])$ still lies in Γ and all the truck movement (up to a reasonable approximation) can be performed in Γ. Otherwise the position variables remain the same and the status variable is set to `error`. When the status variable has the `error` value, the system state remains the same, that is the error is *unrecoverable*. Moreover, after computing the new value of θ_C, we adjust it to satisfy *jackknife* constraint: $|\theta_S - \theta_C| \leq 90°$. See [8] for details.

To simplify the problem of ensuring that the maneuver can be performed inside the parking region Γ we used a Monte Carlo method to estimate a *security border* value (0.98m) to expand the obstacles perimeter. Again, see [8] for details.

4.3 Experimental Results

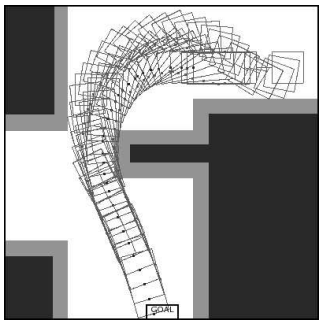


Figure 3. Optimal Trajectory generated by CGMurφ from initial position $x = 12$, $y = 16$, $\theta_S = 0$, $\theta_C = 0$.

We tested our methodology using several obstacles topologies. As an example, we consider the map shown in Figure 3, where the black shapes represent the obstacles and the security borders are drawn grey. In the following we present the results of each phase of our controller generation methodology.

To synthesize the optimal controller we approximate real variables rounding x and y to 0.2 meters and θ_S, θ_C and u to the nearest degree. The resulting controller is generated in 32847 seconds (using a 2.8GHz Intel Xeon workstation with 4GB of RAM) and its table contains 1749586 entries. The state space explored to synthesize the optimal trajectories has 12227989 states, showing how CGMurφ is able to deal with systems having millions of states.

In the second phase, we single out the *safe* states in the optimal controller by calculating the probability p_c on each state (see Sect. 2.2). Using Eq. 1 with $k_c = 4$ and $n = 29$ we obtain the distribution of probability p_c shown in Figure 4. The graph shows the number of states having a given value for p_c. It is clear that most of the states are *safe* (high values of p_c), whereas there is a little but consistent set of *unsafe* states. If we set M_c to 0.1, we have that $|S| = 1493876$ and so in the last phase we have to consider only 85% of states in the optimal controller.

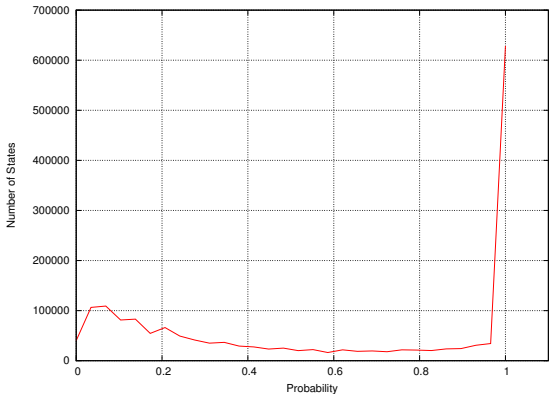


Figure 4. Distribution of p_c on the optimal controller states

Note that, in the system under consideration, there are two kinds of *extreme* positions: (a) when the truck is very near to obstacles and (b) when the truck and trailer are in the *jackknife* position (i.e. when $|\theta_S - \theta_C| = 90°$). Indeed, in the case (a) only a very little number of actions are safe for the truck, whereas the other ones make it crash on an obstacle. On the other hand, in the case (b), it is very difficult to bring the truck outside the jackknife position, since the truck could follow an exceedingly long *almost circular* trajectory. Therefore, according to Section 2.2, this phase correctly identifies as *unsafe* all the states corresponding to positions of case (a). On the other hand, positions of case (b) could be identified as *unsafe* only after the third phase.

Finally, in the last phase we consider the set S generated in the previous phase and we perform the strengthening of the optimal controller using the algorithm of section 2.3 and choosing $k_s = 4$, $M_l = 0.7$ and $M_h = 0.2$ as the constant values.

After the strengthening phase, a significant number of entries (1497114) has been added to the controller to make it robust, due to the complexity of the truck-trailer dynamics. On the other hand, 218260 more states of the controller have been marked as *unsafe*: these states correspond to jackknife positions of the truck and trailer. The final controller now handles a total of 3246700 states.

Note that the last two phases were parallelized by partitioning the controller states in 9 subsets and performing the analysis and strengthening separately for each of these partitions using different workstations. In this way, generating the robust controller took less than an hour.

In order to check the robustness of final controller, we

considered, from each *safe* state in the controller, a trajectory starting from it. For each state s occurring in a given trajectory, we applied a random disturbance on the state variables, generating a new state s^p, and then we applied to s^p the rule associated to the controller state s' that is nearest to s^p. A trajectory is *robust* if, applying the disturbances above, it eventually reaches the goal state.

Disturb. Range for x,y	*Previous* Methodology	
	Disturb. for θ_S,θ_C	Robust Trajectories
$\pm$ 0.1m	$\pm$ 1°	74%
$\pm$ 0.25m	$\pm$ 1°	<70%
$\pm$ 0.5m	$\pm$ 1°	<70%
Disturb. Range for x,y	*New* Methodology	
	Disturb. for θ_S,θ_C	Robust Trajectories
$\pm$ 0.1m	$\pm$ 5°	95%
$\pm$ 0.25m	$\pm$ 5°	94%
$\pm$ 0.5m	$\pm$ 5°	91%

Table 1. Previous and new results about Controller Robustness

As shown in Table 1, we obtain completely satisfying percentages of robust trajectories: even in the presence of big disturbances (0.5 meters for x and y and 5 degrees for θ_s and θ_c) the controller robustness is more than 90%. Moreover, we have a significant improvement (more than 20%) w.r.t. the results of our previous approach described in [8].

5 Conclusions

In this paper we described a probabilistic approach to the robustness problem for numerical controllers. Our methodology exploits the model checking technology to identify the most *unsafe* states of the system, i.e. the untractable ones, thus allowing to focus the controller strengthening only on the others. Indeed, we experimented this methodology on the *truck and trailer with obstacles avoidance* case study, obtaining a controller with a robustness degree of 91-95%, even in the presence of big disturbances. This is a completely satisfactory result for such a complex problem.

This algorithm is now part of our CGMurφ tool [5], which allows to generate robust and optimal controllers through a completely automatic three-step highly parallelizable process.

The tool offers a very versatile product to its potential user. Indeed, thanks to the parallelization, she/he can experiment, in a reasonable time, the controller generation with different safety thresholds until the required controller robustness is obtained. Moreover, she/he can also use the basic optimal controller w.r.t. the *unsafe* states, at the cost of a high precision.

Another point that deserves consideration is the size of our numerical controllers. The corresponding tables contain millions of states and have an average size of tens of Megabytes. So, we are currently working on compression techniques to obtain a reduction of the controller size without affecting its performances.

References

[1] V. I. Arnold. *Ordinary Differential Equations*. Springer-Verlag, Berlin, 1992.

[2] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2005.

[3] F. Borrelli, M. Baotic, A. Bemporad, and M. Morari. Dynamic programming for constrained optimal control of discrete-time linear hybrid systems. *Automatica*, 41:1709–1721, 2005.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[5] CGMurphi Web Page. `http://www.di.univaq.it/magazzeni/cgmurphi.php`.

[6] CMurphi Web Page. `http://www.dsi.uniroma1.it/~tronci/cached.murphi.html`.

[7] E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. McGrawHill, 1955.

[8] G. Della Penna, B. Intrigila, D. Magazzeni, I. Melatti, A. Tofani, and E. Tronci. Automatic generation of optimal controllers through model checking techniques. *to be published in Informatics in Control, Automation and Robotics III, Springer-Verlag (draft available at the url http://www.di.univaq.it/magazzeni/cgmurphi.php*.

[9] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli. Exploiting transition locality in automatic verification of finite state concurrent systems. *STTT*, 6(4):320–341, 2004.

[10] H. Kautz, W. Thomas, and M. Y. Vardi. 05241 executive summary – synthesis and planning. In H. Kautz, W. Thomas, and M. Y. Vardi, editors, *Synthesis and Planning*, number 05241 in Dagstuhl Seminar Proceedings, 2006.

[11] G. Kreisselmeier and T. Birkholzer. Numerical nonlinear regulator design. *IEEE Transactions on Automatic Control*, 39(1):33–46, 1994.

[12] D. Nguyen and B. Widrow. The truck backer-upper: an example of self learning in neural networks. In *In Proceeding of IJCNN.*, volume 2, pages 357–363, 1989.

[13] M. Papa, J. Wood, and S. Shenoi. Evaluating controller robustness using cell mapping. *Fuzzy Sets and Systems*, 121(1):3–12, 2001.