A framework for the competitive analysis of model predictive controllers

Reference:
Bellis Stijn, Denil Joachim, Krishnamurthy Ramesh, Leys Tim, Pérez Guillermo Alberto, Raha Ritam.- A framework for the competitive analysis of model predictive controllers
Lecture notes in computer science - ISSN 1611-3349 - Cham, Springer, 2023, 14235, p. 141-154
Full text (Publisher's DOI): https://doi.org/10.1007/978-3-031-45286-4_11
To cite this reference: https://hdl.handle.net/10067/1997690151162165141

A Framework for the Competitive Analysis of Model Predictive Controllers

Stijn Bellis, Joachim Denil, Ramesh Krishnamurthy, Tim Leys, Guillermo A. Pérez, and Ritam Raha

University of Antwerp – Flanders Make, Belgium
ramesh.krishnamurthy@uantwerpen.be

Abstract. This paper presents a framework for the competitive analysis of Model Predictive Controllers (MPC). Competitive analysis means evaluating the relative performance of the MPC as compared to other controllers. Concretely, we associate the MPC with a regret value which quantifies the maximal difference between its cost and the cost of any alternative controller from a given class. Then, the problem we tackle is that of determining whether the regret value of is at most some given bound. Our contributions are both theoretical as well as practical: (1) We reduce the regret problem for controllers modeled as hybrid automata to the reachability problem for such automata. We propose a reachability-based framework to solve the regret problem. Concretely, (2) we propose a novel CEGAR-like algorithm to train a deep neural network (DNN) to clone the behavior of the MPC. Then, (3) we leverage existing reachability analysis tools capable of handling hybrid automata with DNNs to check bounds on the regret value of the controller.

Keywords: Competitive analysis · Hybrid automata

1 Introduction

An optimal control problem (OCP) deals with finding a function $u(t)$, called a *control law* that assigns values to control variables for every time step $t \in \mathbb{R}_{\geq 0}$. The control law should minimize a given cost function $J[x(\cdot), u(\cdot), t_0, t_f]$ evaluated for a time interval (t_0, t_f) and subject to the state-equation constraints $\dot{x}(t) = f[x(t), u(t), t]$. Model predictive controllers (MPC) solve such a control problem for a given f. This paper presents an approach for the competitive analysis of MPC. Competitive analysis, in this context, means evaluating the relative performance of the MPC as compared to other controllers. Referring to the OCP, our approach assumes that a control law $u(t)$ is given to us. Further, we associate to $u(t)$ a *regret value*, which quantifies the maximal difference between its cost and the cost of any alternative control law from a given class $\mathcal{C}$. Formally, the regret of $u(t)$ is: $\mathit{Reg}(u) := \sup_{c \in \mathcal{C}} \sup_{t_f \in \mathbb{R}_{\geq 0}} J[x(\cdot), u(\cdot), t_0, t_f] - J[x(\cdot), c(\cdot), t_0, t_f]$. If $\mathit{Reg}(u) < r$, then we say that the control law $u(t)$ is r-competitive.

In this work, we first show that the r-competitivity problem for controllers modeled as hybrid automata is interreducible with the reachability problem for hybrid automata. It follows that the r-competitivity problem is undecidable. Fortunately, this also points to using approximate reachability analysis tools to realize approximate competitive analysis. Based on the latter, we propose a counterexample-guided abstraction refinement (CEGAR) framework that abstracts a given MPC using a deep neural network (DNN) trained to clone the behavior of the MPC. This abstraction allows us to use reachability analysis tools such as Verisig [14] to overapproximate the regret value of the abstracted controller. As usual with CEGAR approaches, the refinement step is the main challenge: If the regret is deemed too high (and Verisig finds a real example of this), then this might be due to our abstraction of the controller as a DNN, the overapproximation incurred by the reachability tool, or it might be a real problem with the MPC. In our proposal, when we cannot match the high-regret example to a behavior of the MPC, we use the output of the reachability analysis tool to augment the dataset used for training the DNN.

As a final contribution, we report on a prototype implementation of our CEGAR framework using Verisig. We have used this prototype to analyze MPC for two well-known control problems. While the approach is promising, we conclude that further tooling support is required for the full automation of the framework.

Related work. Chen et al. 2022 [5] conducted a survey on recent advancements in verifying cyber-physical systems and identified as understudied the verification of control systems whose performance is measured using cost functions. Indeed, we did not find many works on the verification of controllers with respect to the cost functions used to obtain them from an OCP instance. Further, to the best of our knowledge, there have been no previous works on the formal analysis of regret in hybrid systems. A notable exception is the recent work of Muvvala et al. [18] who propose regret minimization as a less pessimistic objective for robots involved in collaborations (e.g., with humans), as opposed to a sole emphasis on worst-case optimization. However, their regret analysis focuses on a higher planning level, distinct from the hybrid-dynamics level of the system, making it closer to the work of Hunter et al. [13] rather than the present one.

Behavioral cloning, also known as imitation learning, is a topic of increasing interest within artificial intelligence (see, e.g. [3,19,20]). We do not claim to have a new behavioral cloning algorithm. Rather, we have integrated a data aggregation step into our CEGAR algorithm for the competitive analysis of hybrid automata. Interestingly, contrary to previous uses of DNNs as proxies for MPC [6,14], we have observed that a successful competitive analysis (i.e., the tool says the controller is r-competitive for a small enough r) suggests one can use the DNN instead of the MPC! Although this does not guarantee that the MPC itself is r-competitive, the DNN demonstrates competitiveness. Moreover, evaluating the DNN to compute the control law proves to be relatively efficient.

2 Hybrid Automata and Competitive Analysis

A *hybrid automaton* (HA, for short) is an extension of a finite-state automaton equipped with a finite set of real-valued variables. The values of the variables change *discretely* along the transitions and they do so *continuously*, over time while staying in a state.

Formally a HA is a tuple $(Q, I, T, \Sigma, X, \mathit{jump}, \mathit{flow}, \mathit{inv})$, where:

- Q is a finite set of states and $I \subseteq Q$ is the subset of initial states,
- Σ is a finite alphabet,
- $T \subseteq Q \times \Sigma \times Q$ is a set of transitions, and
- X is a finite set of real-valued variables. We write $V \subseteq \mathbb{R}^X$ to denote the set of all possible valuations of X.
- $\mathit{jump} : T \to \mathit{Op}$ maps transitions to a set of *guards* and *effects* on the values of the variables. That is, $\mathit{Op} \subseteq V^2$ and for a transition $\delta \in T$, $\mathit{jump}(\delta) = (\mathit{guard}, \mathit{effect})$ implies that δ is "enabled" if the current valuation is *guard* and *effect* is the valuation after the transition. Intuitively, *jump* denotes the discrete changes in the variables along transitions. Usually, the guards and effects are encoded as first-order predicates over the reals, e.g. $\mathit{jump}(\delta) = (x > 2, x + 4)$ denotes the set $\{(v, v') \in V^2 \mid v(x) > 2 \text{ and } v'(x) = v(x) + 4\}$.
- $\mathit{flow} : Q \to F$, with $F \subseteq \{f : \mathbb{R}_{>0} \to V\}$, maps each state $q \in Q$ to a set F of functions f_q that give the continuous change in the valuation of the variables while in state q. Usually, the functions f_q are encoded as systems of first-order differential equations, e.g. $\dot{x} = 5$ denotes functions[1] $f(t)(x) = 5t + c$, where $c \in \mathbb{R}_{>0}$ is the value of x at time $t = 0$.
- $\mathit{inv} : Q \to 2^V$ maps each state $q \in Q$ to an invariant that constrains the possible valuations of the variables in q. Similar to *jump*, *inv* is usually encoded as first-order predicates over the reals.

Configurations and runs. A configuration is a pair (q, v) where $q \in Q$ and $v \in V$ is a valuation of the variables in X. A configuration (q, v) is *valid* if $v \in \mathit{inv}(q)$. Let (q, v) and (q', v') be two valid configurations. We say (q', v') is a *discrete successor* of (q, v) if $\delta = (q, a, q') \in T$ for some $a \in \Sigma$ and $(v, v') \in \mathit{jump}(\delta)$. Similarly, (q', v') is a *continuous successor* of (q, v) if $q = q'$ and there exist $t_0, t_1 \in \mathbb{R}_{>0}$ and $f_q \in \mathit{flow}(q)$ such that $f_q(t_0) = v, f_q(t_1) = v'$ and for all $t_0 \le t \le t_1$, $f_q(t) \in \mathit{inv}(q)$.

A *run* ρ is a sequence of configurations $(q_0, v_0)(q_1, v_1) \dots (q_n, v_n)$ such that $q_0 \in I$, v_0 assigns 0 to all variables and, for all $0 \le i < n$, (q_{i+1}, v_{i+1}) is a discrete or continuous successor of (q_i, v_i). The REACH decision problem asks, for given A and (q, v), whether there is a run of A whose last configuration is (q, v).

Parallel composition. Let $A_i = (Q_i, I_i, T_i, \Sigma_i, X_i, \mathit{jump}_i, \mathit{flow}_i, \mathit{inv}_i)$ for $i = 1, 2$ be two HA. Then, $A = (Q, I, T, \Sigma, X, \mathit{jump}, \mathit{flow}, \mathit{inv})$ is the *parallel composition of* A_1 and A_2, written $A = A_1 \mid\mid A_2$, if and only if:

- $Q = Q_1 \times Q_2$ and $I = I_1 \times I_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$ and $X = X_1 \cup X_2$.
- The transition set T contains $(\langle q_1, q_2 \rangle, \sigma, \langle q_1', q_2' \rangle)$ if and only if there are $i, j \in \{1, 2\}$ such that $i \neq j$ and:

[1] Note that if X contains more variables than just x, this function is not unique.

  - either $\sigma \in \Sigma_i \setminus \Sigma_j$, $(q_i, \sigma, q_i') \in T_i$, and $q_j = q_j'$;
  - or $\sigma \in \Sigma_i \cap \Sigma_j$, $(q_i, \sigma, q_i') \in T_i$, and $(q_j, \sigma, q_j') \in T_j$.
- The *jump* function is such that, for $\delta = (\langle q_1, q_2\rangle, \sigma, \langle q_1', q_2'\rangle)$, we have that:
  - either $\sigma \in \Sigma_i \setminus \Sigma_j$ and $\mathit{jump}(\delta) = \mathit{jump}_i(\langle q_i, \sigma, q_i'\rangle)$ for some $i, j \in \{1, 2\}$ with $i \neq j$,
  - or $\sigma \in \Sigma_i \cap \Sigma_j$ and $\mathit{jump}(t) = \mathit{jump}_1(\langle q_1, \sigma, q_1'\rangle) \cap \mathit{jump}_2(\langle q_2, \sigma, q_2'\rangle)$.
- Finally, $\mathit{flow}(\langle q_1, q_2\rangle) = \mathit{flow}_1(q_1) \cap \mathit{flow}_2(q_2)$, and
- $\mathit{inv}(\langle q_1, q_2\rangle) = \mathit{inv}_1(q_1) \cap \mathit{inv}_2(q_2)$.

2.1 The cost of control

In this work, we use HA to model *hybrid systems* and *controllers*. In particular, we henceforth assume any HA $A = (Q, I, T, \Sigma, X, \mathit{jump}, \mathit{flow}, \mathit{inv})$ modelling a hybrid system has a designated *cost* variable $J \in X$. We make no such assumption for HA used to model controllers. Observe that from the definition of parallel composition, it follows that if A models a hybrid system, then $B = A \,||\, C$ also models a hybrid system — i.e. it has the cost variable J — for any HA C.

The following notation will be convenient: For a run $\rho = \dots (q_n, v_n)$ we write J_ρ to denote the value $v_n(J)$. Further, we write $\rho \in A$, where ρ is a run of the hybrid automaton A. Now, the *maximal and minimal cost of a HA A* respectively are $\overline{J(A)} \coloneqq \sup_{\rho \in A} J_\rho$ and,$\underline{J(A)} \coloneqq \inf_{\rho \in A} J_\rho$.

2.2 Regret

Fix a hybrid-system HA $A = (Q, I, T, \Sigma, X, \mathit{jump}, \mathit{flow}, \mathit{inv})$. We define the *(worst-case) regret* $\mathit{Reg}(U)$ of a controller HA U as the maximal difference between the (maximal) cost incurred by the parallel composition of A and U — i.e. the controlled system — and the (minimal) cost incurred by an alternative controller HA from a set $\mathcal{C}$: $\mathit{Reg}(U) \coloneqq \sup_{U' \in \mathcal{C}} (\overline{J(A \,||\, U)} - \underline{J(A \,||\, U')})$. The Regret problem asks, for given A, U, $\mathcal{C}$, and $r \in \mathbb{Q}$, whether $\mathit{Reg}(U) \geq r$.

3 Reachability and Competitive Analysis

In this section, we establish that the reachability and regret problems are interreducible. While this implies an exact algorithm for the competitive analysis of hybrid automata does not exist, it suggests the use of approximation algorithms for reachability as a means to realize an approximate analysis.

Theorem 1. *Let $\mathcal{C}$ be the set of all possible controllers. Then, the* Regret *problem reduces in polynomial time to the* Reach *problem.*

Proof (of Theorem 1). Given a hybrid-system HA A, a controller U, a set of all possible controllers $\mathcal{C}$ and a regret bound $r \in \mathbb{Q}$, we will construct another HA $A' = (Q', I', T', \Sigma, X', \mathit{jump}', \mathit{flow}', \mathit{inv}')$ and a target configuration (q', v) of A' such that, (q', v) is reachable in A' if and only if $\mathit{Reg}(U) < r$ in $A \,||\, U$. Let us write $A = (Q, I, T, \Sigma, X, \mathit{jump}, \mathit{flow}, \mathit{inv})$ and note that $J \in X$ because A is a

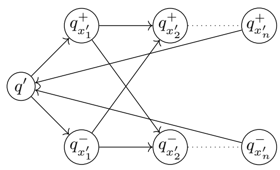


Fig. 1. Gadget for simulating any possible controller

hybrid-system HA. We extend the automaton $A \,||\, U$ with a gadget to obtain A'. The idea is as follows: for every variable $y \in Y$ of $A \,||\, U$, we add a copy of it in the variable set X' of A' that simulates any possible choice of value for that variable by an alternative controller U'. The variable $J' \in X'$ calculates the cost of that alternative controller. Formally, $X' = Y \cup \{y' \mid y \in Y\}$.

To simulate any possible valuation of the variables, we introduce the gadget given in Figure 1. For every variable x'_i such that x_i is a variable in U, the gadget contains two states $q^+_{x'_i}$ and $q^-_{x'_i}$. Then, *flow*$'(q^+_{x'_i})$ contains $\dot{x}'_i = 1$ and $\dot{x}'_j = 0$ for all $j \neq i$. Intuitively, this state allows us to positively update the value of x'_i to any arbitrary value. Similarly, *flow*$'(q^-_{x'_i})$ contains $\dot{x}'_i = -1$ and $\dot{x}'_j = 0$, $\forall j \neq i$, which allows it to negatively update the value of x'_i.

Now, we add a “sink” state q_{reach} and make it reachable from all the other states using transitions $\delta'_i \in T'$ such that *jump*$'(\delta'_i)$ contains guard of the form $J - J' \geq r$. Finally, from every state $q' \in Q'$, we add the option to go into its own copy of the gadget, set the values of the variables to any desired value and come back to the same state.

Note that if $(q_{\text{reach}}, \mathbf{0})$ is reachable in A', via a run $\rho \in A'$, then $J_\rho - J'_\rho \geq r$. As the gadget does not update the value of J and J', it is easy to see that $\mathit{Reg}(U) \geq r$. Now, if $(q_{\text{reach}}, \mathbf{0})$ is not reachable that means, $J_\rho - J'_\rho < r$ for all $\rho \in A'$. Now, as all possible controllers (in fact, all possible configurations of variables from U) can be simulated in A', it is easy to see that $\mathit{Reg}(U) < r$. □

Interestingly, the construction presented above does not preserve the property of being *initialized*. Intuitively, an initialized hybrid automaton is one that “resets” a variable x on transitions between states which have different flows for x. Alas, we do not know whether an alternative proof exists which does preserve the property of being initialized (and also being rectangular, a property which we do not formally define here). Such a reduction would imply the regret problem is decidable for rectangular and initialized hybrid automata.

We now proceed to stating and proving the converse reduction.

Theorem 2. *The* Reach *problem reduces in polynomial time to the* Regret *problem.*

Proof (of Theorem 2). Given a HA A and a target configuration (q, v), we will construct a HA A' and a controller U such that $\mathit{Reg}(U) \geq 2$ with respect to $A' \,||\, U$ if and only if (q, v) is reachable in A. The reduction works for any set $\mathcal{C}$ of controllers that contains at least one controller that sets c to 0 all the time.

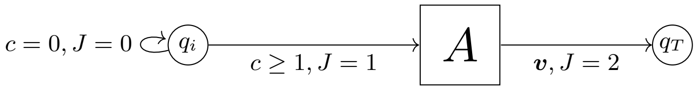


Fig. 2. Reduction from REACH to REGRET

First, we add two states to A' so that $Q' = Q \cup \{q_i, q_T\}$. In A', q_i has a self-loop that can be taken if the value of c is 0 and the effect is that $J = 0$. From q_i, we can also transition to the initial states of A if $c \geq 1$, and in doing so, we set J to 1. Finally, from the target state q in A, we can go to the new state q_T if the target valuation $\boldsymbol{v}$ is reached, and that changes the valuation of J to 2. The valuation of J does not change within A.

Note that the minimum cost incurred by a controller that constantly sets c to 0 in A' is 0, which is achieved by the run that loops on q_i. Now, if $(q, \boldsymbol{v})$ is reachable in A via run $\rho \in A$, then the maximum cost incurred by a controller that sets c to 1 occurs along a run $q_i \cdot \rho \cdot q_T$ and is 2, making $Reg(U) \geq 2$. On the other hand, if $(q, \boldsymbol{v})$ is not reachable in A, then the maximal value of J along any such run is 1, resulting in $Reg(U) < 2$. Our constructed controller U is such that it sets c to 1 all the time, and the above arguments give the desired result. □

Since the reachability problem is known to be undecidable for hybrid automata in general [11], it follows that our regret problem is also undecidable.

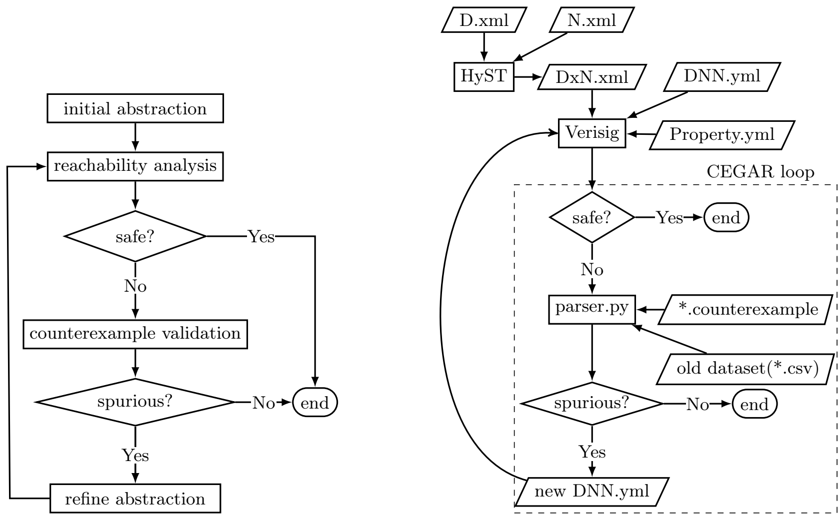
Corollary 1. *The* REGRET *problem is undecidable.*

4 CEGAR-Based Competitive Analysis

We present our CEGAR approach to realize approximate competitive analysis. To keep the discussion simple, we focus on continuous systems, specifically single-state hybrid automata. Since our goal is to approximate the regret of MPCs, we model controllers as hybrid automata that sample variable values at discrete-time intervals and determine control variable values using a deep neural network (DNN) trained to behave as the MPC. Concretely, our approach specializes the reduction in the proof of Theorem 1: We will work with a hybrid automaton $\mathcal{D}$ that abstracts the behavior of the controller using a DNN, and a hybrid automaton $\mathcal{N}$ that abstracts the behaviors of all alternative controllers. The overview of our framework is depicted in Figure 3a. In section 5, we present a toolchain implementing this CEGAR-based approach.

4.1 Initial abstraction and analysis

Our proposed framework begins with the abstraction of the controller as a hybrid automaton $\mathcal{D}$ and the alternative controllers as $\mathcal{N}$. Each of these automata are assumed to have a cost variable, say $J_{\mathcal{D}}$ for $\mathcal{D}$ and $J_{\mathcal{N}}$ for $\mathcal{N}$. For a given



(a) CEGAR-based approach (b) Competitive analysis toolchain

Fig. 3. Flowchart depictions of our approach and our toolchain implementing it; We use ANSI/ISO standard flowchart symbols: the parallelogram blocks represent inputs/outputs, and the rectangular blocks represent processes or tools

value $r \in \mathbb{R}$, if we want to determine whether $\mathcal{D}$ is r-competitive then we add to $\mathcal{A} = \mathcal{D}||\mathcal{N}$ a new cost variable $J = J_{\mathcal{N}} - J_{\mathcal{D}}$. As is argued in Theorem 1, $\mathcal{D}$ should be r-competitive if and only if $\mathcal{A}$ can reach a configuration where the value of J is larger than r. Hence, we can apply any reachability set (overapproximation) tool to determine the feasibility of such a configuration.

4.2 Reachability status

If the reachability tool finds that a configuration with $J \geq r$ is reachable in $\mathcal{A}$, we say it concludes $\mathcal{A}$ is *unsafe*. In that case, we will have to process the reachability witness. Otherwise, $\mathcal{A}$ is *safe*, and we can stop and conclude that $\mathcal{D}$ is r-competitive. Interestingly, $\mathcal{D}$ can now be used as an r-competitive replacement of the original controller! It is important to highlight that behavior cloning does not provide any guarantees regarding the relationship between the MPC and the DNN within $\mathcal{D}$. Consequently, even if we have evidence supporting the r-competitiveness of $\mathcal{D}$, we cannot infer the same for the MPC itself.

In the context of MPCs, this result is already quite useful. This is because MPCs have a non-trivial *latency* and memory usage before choosing a next valuation for the control variables (see, e.g. [12,15]). In our implementation described in the following section, $\mathcal{D}$ takes the form of a DNN. As DNNs can be evaluated rather efficiently, using the DNN instead of the original MPC is desirable.

4.3 Counterexample analysis and refinement

If $\mathcal{A}$ is deemed unsafe, we expect the reachability tool to output a counterexample in the form of a run. There is one main reason why such a run could be *spurious*, i.e. it does not correspond to a witness of the MPC not being r-competitive. Namely, the abstractions $\mathcal{D}$ (representing the MPC) or $\mathcal{N}$ (representing alternative controllers) might be too coarse. For the specific case of $\mathcal{D}$, where a DNN is used to model the MPC, we describe sufficient conditions to determine if the counterexample is indeed spurious. If the counterexample is indeed deemed spurious, we can refine our abstraction by incorporating new data obtained from the counterexample and retraining the DNN. In general, though, refining $\mathcal{D}$ and $\mathcal{N}$ falls into one of the tasks for which our framework does not rely on automation.

4.4 Human in the loop

There are three points in the framework, where human intervention is needed.

Modelling and specification. First, the task of obtaining initial abstractions $\mathcal{D}$ and $\mathcal{N}$ of the controller and all alternative controllers, respectively, does require a human in the loop. Indeed, crafting hybrid automata is not something we expect from every control engineer. In our prototype described in the next section, we mention partial support for obtaining $\mathcal{D}$ and $\mathcal{N}$ automatically when the MPC is given in the language of a particular OCP and optimization library.

Reachability analysis. Second, reachability being an undecidable problem, most reachability analysis tools can not only output safe and unsafe as results. Additionally, they might output an "unknown" status. In this case, revisiting the abstractions $\mathcal{D}$ and $\mathcal{N}$, or even changing the options with which the tool is being used may require human intervention. In fact, we see this as an additional abstraction-refinement step which is considerably harder to automate since there is an absence of a counterexample to work with.

Abstraction refinement. Finally, our framework does not say what to do if the counterexample being spurious is due to $\mathcal{N}$ being too coarse an approximation. This scenario can occur when $\mathcal{N}$ is purposefully modeled to discretize or approximate certain behaviors of alternative controllers to facilitate reachability analysis. However, for $\mathcal{D}$, we offer automation support by proposing the retraining of our DNN in the implementation. It might actually be needed to change the architecture of the DNN to obtain a better abstraction. This process can be automated, as increasing the number of layers is often sufficient according to the universal approximation theorem [4].

5 Implementation and Evaluation

We now present our implementation of the CEGAR-based competitive analysis method presented in the previous section, along with two case studies used for evaluation: the cart pendulum and an instance of motion planning.

5.1 Competitive analysis toolchain

Figure 3b gives a visual depiction of the toolchain in the form of a flowchart. Starting from the top, `D.xml`, `N.xml` are XML files encoding hybrid automata $\mathcal{D}$ and $\mathcal{N}$, respectively, in the SpaceEx modeling language[8]. The automaton $\mathcal{D}$ represents the controller, which could be a model predictive controller (MPC), and $\mathcal{N}$ represents a class of controllers that the MPC is compared against — see also Section 4.1. We use the HyST [2] translation tool for hybrid automata to generate the parallel composition $\mathcal{D} \,||\, \mathcal{N}$ (encoded in `DxN.xml`, again in the SpaceEx language). The composed automaton, along with the trained DNN and the property to be verified, are fed as inputs to Verisig. Verisig [14] is a tool that verifies the safety properties of closed-loop systems with neural network components. The tool takes a hybrid automaton, a trained neural network, and property specification files as inputs. It performs the reachability analysis and provides safety verification result. We then parse the output of Verisig to determine whether D is competitive enough (`parser.py`). If this is not the case, we realize a sound check to determine if the counterexample is spurious, in which case we use it to extend our dataset and further train the DNN.

5.2 Initial abstraction and training

Our toolchain is finetuned to work well for hybrid systems modeled in a tool called *Rockit* and MPCs obtained using the same tool. Rockit, which stands for Rapid Optimal Control Kit, is a tool designed to facilitate the rapid prototyping of optimal control problems, including iterative learning, model predictive control, system identification, and motion planning [10].

Our toolchain includes a utility that interfaces with the API of Rockit to automatically generate the hybrid automata $\mathcal{D}$ and $\mathcal{N}$ from a model of a control problem. While the use of Rockit is convenient, it is not required by our toolchain.

Based on a dataset (in our examples, we obtain it from Rockit), we train a DNN using *behavioral cloning*: we try to learn the behavior of an *expert* (in our case, the MPC) and replicate it. For this, we make use of the *Dagger algorithm*[20], which, after an initial round of training on the dataset from Rockit, will simulate traces using the DNN. The points that the neural network visits along these traces are then given to the expert, and the output of the expert is recorded. These new points and outputs are appended to the first dataset, and this new dataset is used to train a second DNN. This iterative process is done multiple times to make the DNN more robust. In all of our experiments, the TensorFlow framework [1] was used for the creation and training of the DNN.

5.3 Reachability status

The regret property, encoded as a reachability property as is done in the proof of Theorem 1, is specified in the property file `Property.yml`, which also includes the initial states of $\mathcal{D} \,||\, \mathcal{N}$. Verisig provides three possible results: "safe" if no property violation is found, "unsafe" if there is a violation, and "unknown" if the property could not be verified, potentially due to a significant approximation error. In the latter two cases, a counterexample file (CE file) is generated.

5.4 Counterexample analysis and retraining

If the result is "unsafe", the next step is to compare the counterexample trajectory against the dataset generated from the controller code. If a matching trajectory is found, it indicates a real counterexample, meaning that this trajectory could potentially occur in the actual controller, and no further action is required. If a matching trajectory is not found, then it is a spurious counterexample that requires either retraining the DNN or fix(es) in $\mathcal{D}\,||\,\mathcal{N}$. Our toolchain automatically validates the counterexample by comparing the trajectories from Verisig and the controller as implemented in Rockit. To do so, since Rockit uses the floating-point representation of real numbers, we choose a decimal precision of $\epsilon = 10^{-3}$ for the comparison. In the case of a spurious counterexample that requires retraining the DNN, we update the existing dataset using Rockit to obtain additional labeled data based on the trajectory from the CE file.

The CE file from Verisig represents state variable values using interval arithmetic, while the controller dataset contains state variable values in $\mathbb{R}$ without intervals. To accommodate this difference, we choose to append to the dataset new entries: (a) the lower bounds of input intervals, (b) the upper bounds, and (c) a range[2] of intermediate input values within the intervals. For each of these, we also include the corresponding controller outputs. The generation of the updated dataset and the retraining of the DNN are performed automatically by our toolchain. A DNN trained on the new dataset is then fed to Verisig again along with `DxN.xml` and the `Property.yml`. This way, the CEGAR loop is repeated until one of the following conditions is true: (a) the counterexample is real, or (b) a maximum number of retraining iterations (determined by the user) is reached.

5.5 Experiments

In the sequel, we use our tool to analyze two control problems that have been implemented using the Rockit framework. The research questions we want to answer with the forthcoming empirical study are the following.

RQ1 Can we have a fully automated tool to perform the competitive analysis?
RQ2 Is the toolchain scalable? Why or why not?
RQ3 Does the approach help to improve confidence in (finite-horizon) competitivity of controllers?
RQ4 Does the approach help find bugs in controller design?

We now briefly introduce the two case studies, their dynamics, and how each of them are modeled so that our toolchain can be used to analyze them.

[2] Our toolchain splits each interval into n equally large segments and adds all points in the resulting lattice. In our experiments, we use $n = 4$.

Cart pendulum. The cart pendulum problem is a classic challenge in control theory and dynamics [7]. In it, an inverted pendulum is mounted on a cart that can move horizontally via an electronic servo system. The objective is to minimize a cost $J = F^2 + 100 * pos^2$, where F represents the force applied to the cart and pos indicates the position of the cart. The values of F and pos are constrained within the range of $[-2, 2]$. The dynamics of the cart correspond to the physics of the system and make use of parameters including the mass of the cart and the pendulum and the length of the pendulum (see Appendix A).

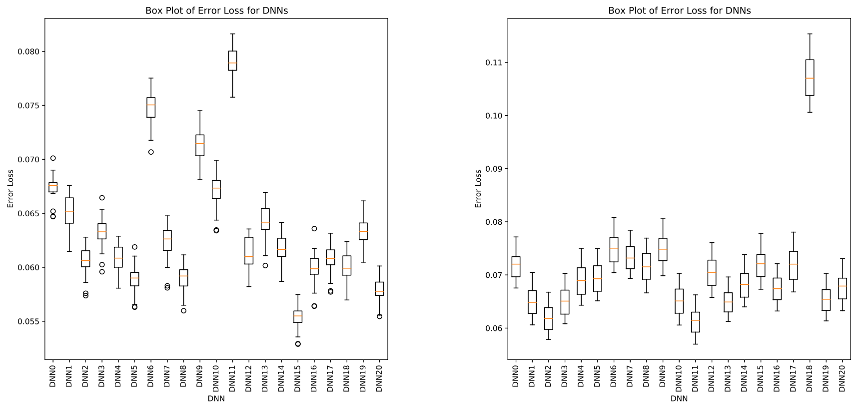
While the proof of Theorem 1 provides a sound way to model all alternative controllers in the form of $\mathcal{N}$, the construction combines continuous dynamics and non-determinism. Current hybrid automata tools do not handle non-trivial combinations of these two elements very well. Hence, we have opted to discretize the choice of control values for alternative controllers. Intuitively, this means that every time the DNN is asked for new control variable values in $\mathcal{D}$, the automaton $\mathcal{N}$ non-deterministically chooses new alternative values from a finite subset fixed by us a priori (see Figure 5b in appendix for an example).

Motion planning The case study involves computing a series of actions to move an object from one point to another while satisfying specific constraints [16]. In our case study, an MPC is used to plan the motion of an autonomous bicycle that is expected to move along a curved path on a 2D plane using a predefined set of waypoints. To prevent high-speed and skidding, the velocity (V) and the turning rate (δ, in radians) are constrained in the ranges $0 \leq V \leq 1$ and $-\pi/6 \leq \delta \leq \pi/6$. The objective is to minimize the sum of squared estimate of errors between the actual path taken by the bicycle and the reference path. Intuitively, the more the controller deviates from the reference path, the higher its cost (see Appendix B).

Like in the cart pendulum case study, we discretize the alternative control variable valuations. A big difference is that the cost has both a *Mayer term* and a *Lagrangian* that depend on the location of the bicycle and the waypoints in an intricate way. In terms of modelling, this means that $\mathcal{D}$ and $\mathcal{N}$ have to "compute" closest waypoints relative to the current position of the bicycle (see Figure 6).

Discussion. Towards an answer for **RQ1**, we can say that while our toolchain[3] somewhat automates our CEGAR, it still requires manual work (e.g. the initial training and choice of DNN architecture). Moreover, in the described case studies, we did not observe an MPC DNN that is labeled as competitive. This may be due to (over)approximations incurred by our framework and our use of Verisig. Despite this, we can answer **RQ4** positively as our toolchain allowed us to spot a bug hidden in the Rockit MPC solution for the cart pendulum. We observed in early experiments that the MPC was not competitive and short (run) examples of this were quickly found by Verisig. We then found that the objective function in Rockit was indeed not as intended by the developers.

[3] All graphs and numbers can be reproduced using scripts from: `https://github.com/competitive-analysis-toolchain/competitive-analysis`.



(a) Cart pendulum (b) Motion planning

Fig. 4. Boxplots showing the training losses of all DNNs against all test sets

The DNNs do show a trend towards copying the behavior of the MPC (see Figure 4) even though we retrain a new DNN from scratch after each (spurious) counterexample obtained via Verisig and we (purposefully) randomize the choice of test and training set in each iteration. We do this to increase variability in the set of behaviors and the counterexamples used to extend the dataset. In the cart pendulum case study, we observe that in the iterations 2, 7, and 11, the number of discrete time steps during which the corresponding DNN can act while remaining competitive is larger than in the initial iteration. Hence, for **RQ3**, we conclude our toolchain can indeed help increase reliability in the DNN proxy being competitive, albeit only for a finite horizon. On the negative side, experiments for 20 iterations of retraining from spurious counterexamples take more than 90min in both our case studies. This leads us to conclude that our toolchain does not yet scale as required for industrial-size case studies (**RQ2**).

6 Conclusion

Based on our theoretical developments to link the regret problem with the classical reachability problem, we proposed a CEGAR-based approach to realize the competitive analysis of MPCs via neural networks as proxies. We also presented an early proof-of-concept implementation of the approach. Now that we have a baseline, we strongly believe improvements in the form of algorithms and dedicated tools will allow us to improve our framework to the point where it scales for interesting classes of hybrid systems.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P.A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: Keeton, K., Roscoe, T. (eds.) 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016. pp. 265–283. USENIX Association (2016), `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`
2. Bak, S., Bogomolov, S., Johnson, T.T.: Hyst: a source transformation and translation tool for hybrid automaton models. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control. pp. 128–133 (2015)
3. Bratko, I., Urbančič, T., Sammut, C.: Behavioural cloning: phenomena, results and problems. IFAC Proceedings Volumes **28**(21), 143–149 (1995)
4. Chen, T., Chen, H.: Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Transactions on Neural Networks **6**(4), 911–917 (1995)
5. Chen, X., Sankaranarayanan, S.: Reachability analysis for cyber-physical systems: Are we there yet? In: NASA Formal Methods Symposium. pp. 109–130. Springer (2022)
6. Clavière, A., Dutta, S., Sankaranarayanan, S.: Trajectory tracking control for robotic vehicles using counterexample guided training of neural networks. In: Benton, J., Lipovetzky, N., Onaindia, E., Smith, D.E., Srivastava, S. (eds.) Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019. pp. 680–688. AAAI Press (2019), `https://ojs.aaai.org/index.php/ICAPS/article/view/3555`
7. Fantoni, I., Lozano, R., Lozano, R.: Non-linear control for underactuated mechanical systems. Springer Science & Business Media (2002)
8. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23. pp. 379–395. Springer (2011)
9. Gillis, J.: rockit. `https://gitlab.kuleuven.be/meco-software/rockit/-/blob/master/examples/motion_planning_MPC.py` (2020)
10. Gillis, J., Vandewal, B., Pipeleers, G., Swevers, J.: Effortless modeling of optimal control problems with rockit. In: 39th Benelux Meeting on Systems and Control, Date: 2020/03/10-2020/03/12, Location: Elspeet, The Netherlands (2020)
11. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998). https://doi.org/10.1006/jcss.1998.1581, `https://doi.org/10.1006/jcss.1998.1581`
12. Hertneck, M., Köhler, J., Trimpe, S., Allgöwer, F.: Learning an approximate model predictive controller with guarantees. IEEE Control. Syst. Lett. **2**(3), 543–548 (2018). https://doi.org/10.1109/LCSYS.2018.2843682, `https://doi.org/10.1109/LCSYS.2018.2843682`
13. Hunter, P., Pérez, G.A., Raskin, J.: Reactive synthesis without regret. Acta Informatica **54**(1), 3–39 (2017). https://doi.org/10.1007/s00236-016-0268-z, `https://doi.org/10.1007/s00236-016-0268-z`

14. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G., Lee, I.: Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In: International Conference on Computer Aided Verification. pp. 249–262. Springer (2021)
15. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC). pp. 1–10. IEEE (2016)
16. LaValle, S.M.: Planning algorithms. Cambridge university press (2006)
17. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research **18**(185), 1–52 (2018), `http://jmlr.org/papers/v18/16-558.html`
18. Muvvala, K., Amorese, P., Lahijanian, M.: Let's collaborate: Regret-based reactive synthesis for robotic manipulation. In: 2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022. pp. 4340–4346. IEEE (2022). https://doi.org/10.1109/ICRA46639.2022.9812298, `https://doi.org/10.1109/ICRA46639.2022.9812298`
19. Ross, S., Bagnell, D.: Efficient reductions for imitation learning. In: Teh, Y.W., Titterington, D.M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010. JMLR Proceedings, vol. 9, pp. 661–668. JMLR.org (2010), `http://proceedings.mlr.press/v9/ross10a.html`
20. Ross, S., Gordon, G.J., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Gordon, G.J., Dunson, D.B., Dudík, M. (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011. JMLR Proceedings, vol. 15, pp. 627–635. JMLR.org (2011), `http://proceedings.mlr.press/v15/ross11a/ross11a.pdf`

A Cart pendulum with physics-based cost

The cart's dynamics are summarized in Table 1, with parameters including the cart's mass ($mcart$), the pendulum mass (m), the pendulum length (L), and the gravitational constant (g).

Use case	**Initial state**	**Dynamics**	**Objective**	**Control variable**	**Path constraints**	**Sample time(s)**
cart pendulum	$pos = 0.5$ $\theta = 0$ $\dot{pos} = 0$ $\dot{\theta} = 0$	$\dot{pos} = dpos$ $\dot{\theta} = d\theta$ $\dot{dpos} = (-mLsin(\theta)(d\theta)^2 +$ $mgcos(\theta)sin(\theta) + F)/$ $(mcart + m - m * (cos(\theta))^2)$ $\dot{d\theta} = (-mLcos(\theta)sin(\theta)(d\theta)^2 +$ $Fcos(\theta) + (mcart + m)gsin(\theta))/$ $(L(mcart + m - m(cos(\theta))^2))$	$2F + 100pos^2$	Force (F)	$-2 \leq F \leq 2$ $-2 \leq pos \leq 2$	0.04
motion planning	$x = 0$ $y = 10$ $\theta = 0$	$x = Vcos(\theta)$ $y = Vsin(\theta)$ $\theta = V/Ltan(\delta)$	square error between position and reference path	Turning rate (δ)	$0 \leq V \leq 1$ $-\pi/6 \leq \delta \leq \pi/6$	$1.3 \leq t \leq 1.9$

Table 1. The initial states, the dynamics and other control parameters of the cart pendulum and motion planning.

To determine the sample time for the MPC, we calculate it as the ratio of the control horizon (Tf) to the number of control intervals ($Nhor$). In this case, Tf is set to 2 seconds and $Nhor$ is 50, resulting in a sample time of $dt = Tf/Nhor = 0.04s$. Additionally, the initial conditions for the system are specified as $[pos, \theta, dpos, d\theta] = [0.5, 0, 0, 0]$. The controller is said to have found an optimal solution, respecting the constraints, for which the cost is minimum. That is, the more the controller deviates from the set constraints for F and pos, the more it is "punished", thereby setting a higher value to the cost variable.

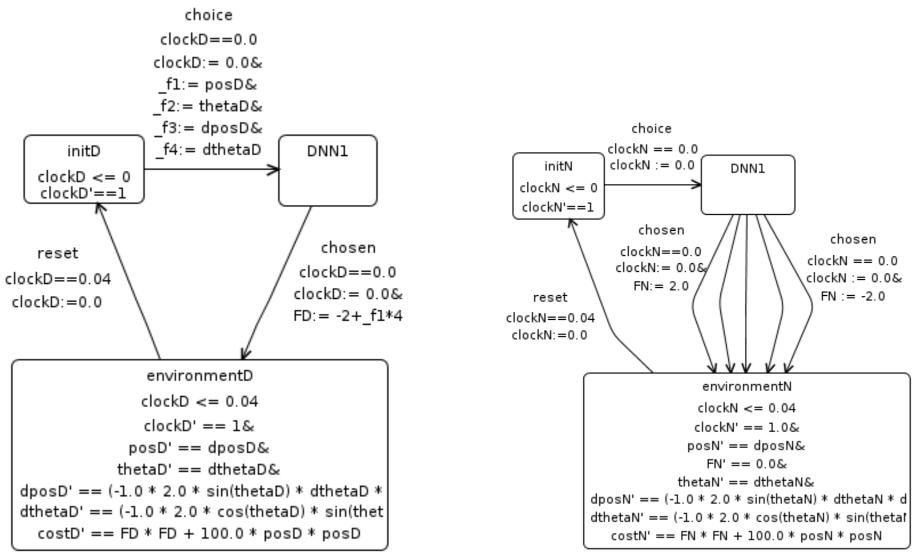
For the DNN abstracting the MPC, we trained a fully connected model with 4 inputs nodes, 1 output node[4] and 4 hidden layers. The number of nodes in the hidden layers as well as the activation function and the learning rate were chosen using hypertuning. For hypertuning, the hyperband algorithm was used [17]. The options that the hyperband algorithm had for each hyperparameter were the following:

- The amount of nodes in each hidden layer: 16, 32, 48, 64, 80 or 96.
- The learning rate was sampled between 1e-5 and 1e-1 with a logarithmic sampling.
- The activation function: sigmoid or a tangent hyperbolic.

The sigmoid and tangent hyperbolic activation functions were chosen because they can be used in Verisig. The factor used in the hyperband algorithm was 3.

[4] The last layer of the DNN has a shifted and scaled sigmoid function as an activation function. This function restricts the output of the DNN between -2 and 2. This makes sure that the DNN never outputs a parameter that is outside the bounds of the problem. This reduces the number of spurious counterexamples that are found by the CEGAR loop.

We begin the modeling process by representing the given controller as automaton $\mathcal{D}$, depicted in Figure Figure 5a. This automaton consists of three modes, each suffixed with a 'D' (e.g., initD, environmentD), distinguishing them from the modes of automaton $\mathcal{N}$. The initial mode, initD, and the DNN mode do not involve any time elapse. The incoming edges to the DNN mode labeled as $\_f1$, $\_f2$, etc., indicate the inputs to the DNN. Verisig uses the input `DNN.yml` in place of the DNN mode. The DNN "mimics" the MPC which takes pos, θ, $dpos$, and $d\theta$ as inputs and outputs F. The output of the DNN ($\_f1$) is set to the Force variable FD with a scaling factor that corresponds to the sigmoid activation function.



(a) Automaton $\mathcal{D}$ (b) Automaton $\mathcal{N}$

Fig. 5. Automaton $\mathcal{D}$ and automaton $\mathcal{N}$ for cart pendulum

The environment mode contains the control dynamics and the cost function, expressed as ordinary differential equations. To model the controller's sample time, we employ a combination of setting the invariant of clockD to 0.04 and resetting clockD to 0.04 on the outgoing edge from the environment. This ensures that exactly 0.04 seconds are spent in the environment mode, reflecting the desired sample time of the controller.

Automaton $\mathcal{N}$ models the class of controllers to be compared with MPC (or alike). Unlike $\mathcal{D}$, $\mathcal{N}$ does not use a trained DNN – as $\mathcal{N}$ corresponds to a class of controllers other than the MPC (or alike), there is no need to have a DNN. So, the DNN mode in $\mathcal{N}$ is just a dummy mode used for sync and composition generation. Also, unlike $\mathcal{D}$, $\mathcal{N}$ has discrete behaviour with deterministic transitions where each transition has F set between -2 and 2 (same constraint as in

$\mathcal{D}$). That is, by discretizing the environment in automaton $\mathcal{N}$ we make the set of controllers we compare against as finite. Furthermore, the discretization helps in having a tighter reachability set, thereby reducing verification time. We assume $\mathcal{D}$ and $\mathcal{N}$ have the same sampling time of 0.04s. Note that the behaviour of $\mathcal{D}$ and $\mathcal{N}$ is identical, for e.g., a transition in $\mathcal{D}$ from init mode to DNN mode is identical to the transition from init mode to DNN mode in $\mathcal{N}$. This ensures a fair comparison between the MPC and other controllers. Synchronization labels (choice, chosen, reset) are used on each edge to establish this identical behaviour, which also facilitates composition. The parallel composition $\mathcal{D}||\mathcal{N}$ generated by HyST.

The next step is to feed the generated composition $\mathcal{D}||\mathcal{N}$ to Verisig along with the property file and the trained DNN. The competitive analysis is performed for the regret property $costD - costN >= 0.25$. That is, we ask Verisig if there is an 'unsafe' trajectory in the composed automaton with the difference in the costs is greater than 0.25. Verisig returned an 'unsafe' result along with a CE file, which is then automatically parsed to obtain the unsafe trajectory. The trajectory from Verisig is compared against the controller's dataset and it is found that it is a spurious counterexample. As the next step, the dataset file is appended with new data from the CE file, and the DNN is retrained. The retrained DNN, along with the composed automaton and the property file is fed to Verisig again. The DNN retraining (a.k.a the refinement) is repeated for 20 iterations.

B Motion planning with waypoint-based cost

Motion planning involves computing a series of optimal steps to move an object from one point to another while satisfying specific constraints [16]. In our case study, an MPC is used to plan the motion of an autonomous bicycle that is expected to move on a curved path on a 2D plane using a predefined set of waypoints. To prevent high speed and skidding, the velocity(v) and the turning rate(δ) of the bicycle are constrained within the ranges of $0 \leq V \leq 1$ and $-\pi/6 \leq \delta \leq \pi/6$ respectively. The objective is to minimize the sum of squares error between the actual path taken by the bicycle and the reference path. That is, the more the controller deviates from the reference path, the more it is "punished" with a higher value of sum squared error. The bicycle dynamics are summarized in Table 1, where L represents the bicycle length. Unlike the cart pole case study where the sample time remains constant, the controller dynamically calculates the sample time (t_s second) within the range of $1.3 <= t_s <= 1.9$. The Python implementation of the use case is available at [9].

To model this case study, four automaton were employed. $\mathcal{D}$ and $\mathcal{N}$ represent the given controller and the class of controllers for comparison, respectively. Additionally, we use two more automaton $\mathcal{C}1$ and $\mathcal{C}2$, one for the given controller and one for the class of controllers. Since these two automaton are identical, automata $\mathcal{C}1$ is shown in Figure 6. These automata model three functions from the path planning algorithm that compare the bicycle's current position with the reference path. The first function involves finding the closest waypoint on

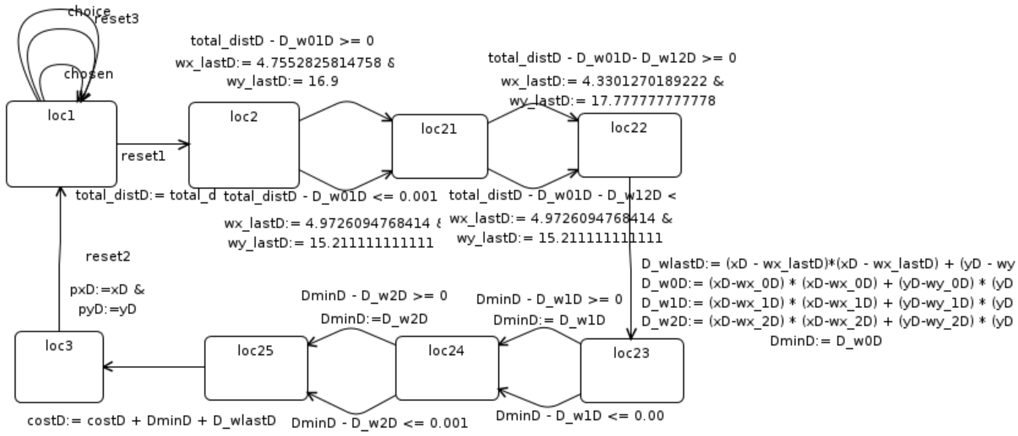


Fig. 6. Automaton $\mathcal{C}1$ for motion planning

the reference path and comparing it with the current position. This is achieved by computing the x- and y-coordinate distances between the current position and each point on the reference path, starting from a specific index. The index of the closest waypoint is saved using the distance formula[5].The second function determines the index of the last waypoint on the reference path that is located at a certain distance from the current position. The last index is obtained by calculating the cumulative distance starting from the index of the first waypoint and continuing until the index of the last waypoint. The third function creates a list of N waypoints. It begins by using the index of the last waypoint obtained from the second step to determine the number of indices (referred to as $delta\_index$) between the last index and the first index. If $delta\_index$ is greater than the desired number of waypoints N, it means there are more than N path points available. In this case, we consider the indices of the first N waypoints. However, if $delta\_index$ is less than N, indicating that there are fewer than N waypoints available, we consider the index of the last waypoint and repeat it multiple times.

In contrast to the physics-based cost utilized in the cart pole case study, the cost in $\mathcal{C}1$ and $\mathcal{C}2$ is modeled as the sum of the accumulated cost, the distance to the closest waypoint, and the distance to the last waypoint. This cumulative sum represents the deviation between the path taken by the controller and the reference path. Subsequently, we conduct a competitive analysis, as illustrated in Figure 3b, utilizing the costs obtained from $\mathcal{C}1$ and $\mathcal{C}2$.

For the DNN abstracting this MPC, we trained a fully connected model with 3 inputs nodes, 3 output node and 4 hidden layers. The 3 input nodes correspond to the parameters X, Y and theta that are used in the MPC. The output nodes

[5] We employ the squared difference between the x- and y-coordinates without taking the square root. This choice is made due to the current limitation of flow*, which does not support the square root of variables (only the square root of a constant is supported).

correspond with the control variables Delta, V and a timestep. The timestep is used by the simulation function of the system because the timestep is not constant for this MPC. This MPC does not use an activation function on the last layer to restrict the output. The rest of the construction and training of this DNN is analogous to the previous DNN.

The generated composition $\mathcal{D} \mid\mid \mathcal{N} \mid\mid \mathcal{C}1 \mid\mid \mathcal{C}2$ is fed to Verisig along with the property file and the trained DNN. The competitive analysis is performed for the regret property $costD - costN >= 0.25$. Verisig returned an 'unsafe' result along with a CE file. The trajectory from Verisig is compared against the controller's dataset and it is found that it is a spurious counterexample. As the next step, the dataset file is appended with new data from the CE file, and the DNN is retrained. The retrained DNN, along with the composed automaton and the property file is fed to Verisig again. The DNN retraining is repeated for 20 iterations.