Self-Organizing Agents to Learn the Control of Heat Engines

Jérémy Boes, Frédéric Migeon and François Gatto

IRIT, University of Toulouse
{boes, migeon, gatto}@irit.fr

Abstract. Controlling heat engines imposes to deal with high dynamics, non-linearity and multiple interdependencies. A way handle these difficulties is enable the controller to learn how the engine behaves, hence avoiding the costly use of an explicit model of the process. Adaptive Multi-Agent Systems (AMAS) are able to learn and to adapt themselves to their environment thanks to the cooperative self-organization of their agents. A change in the organization of the agents results in a change of the emergent function. Thus we assume that AMAS are a good alternative for complex systems control, reuniting learning, adaptivity, robustness and genericity. In this paper, we present an AMAS for the control of heat engines and show several results.

1 INTRODUCTION

Over the years, the complexity of car engines has been growing up, with new sophistications like Exhaust Gas Recirculation or particle filters. Moreover our expectations about their efficiency are becoming more important due to the strengthening of environmental norms, while the need for economic competitivity leads to a need for easily instantiable controllers. The emergence of software and electronic devices helps to tackle these challenges but the development of Engine Control Unit (ECU) remains a difficult and time consuming task.

The goal of this paper is to introduce a non-classical approach of the control of car engines, based on Adaptive Multi-Agent Systems (AMAS), that aims to provide a quickly instantiable controller, avoiding a heavy parametrization work. Based on learning and adaptiveness skills of AMASs, our approach considers the engine and its ECU as a black box. The controller modifies ECU parameters in order to obtain the desired values on the engine outputs. This work is part of a national project named Orianne.

Our motivation for working on a non-classical approach comes from the difficulties to build and fit an efficient model of the engine we intend to control. Humans fulfill this control task in their everyday life. A child learning to drive a bike is a good example. He has the ability to move his legs in order to walk and run. He does not have any knowledge about bike gears or inertia theory. While he is experiencing his first ride, his most used skill is the ability to adapt and to coordinate his movements in a new situation. This adaptation requires learning, which is certainly not an instantaneous process. Following a practice period, the child eventually ends up acquiring the ability to control the bike but still did not get in touch with any theoretical concept and did not build

any analytical model of the bike inside his head. Furthermore, he never stops learning: each new ride will improve his mastering of the bike, and he will be able to adapt to a new bike. This analogy is meant to explain the point of view we adopted: the use of an explicit analytical model is not necessary to the control of an engine.

Section 2 introduces usual control methods. We follow with an introduction to self-organization and to the Adaptive Multi-Agent Systems (AMAS) theory. Section 3 describes the design of a Multi-Agent System before going deeper in the agents behavior in section 4. Then section 5 shows some results, before section 6 concludes with our perspectives.

2 RELATED WORKS

In this section the main approaches of control are presented before self-organization and the Adaptive Multi-Agent Systems theory are briefly introduced.

2.1 Complex Systems Control

Controlling systems is a generic problem that can be expressed as finding which modifications are needed to be applied on the inputs in order to obtain the desired effects on the ouputs. The main approaches are presented in the next paragraphs.

PID Proportional-Integral-Derivative (PID) controllers are well-known and widely used. They base the control on the calculation and the combination of three terms, corresponding to the error (P), its duration (I) and its variation rate (D) [1]. The instantiation of a PID controller to a particular system is a heavy procedure despite adjustement methods like [19]. Besides, PID controllers are not always efficient with non-linear systems, they need to be complemented with other mechanisms like fuzzy logic [21]. Also, a PID controller can only handle one input with one retroaction signal, which is a severe drawback for complex systems control.

Adaptive Control The dynamic nature of complex systems imposes to adapt the controller as the system evolves. Model-based approaches like Model Identification Adaptive Control (MIAC) [17] or Model Reference Adaptive Control (MRAC) [11] are regrouped under the name of Model Predictive Control (MPC) [15]. A model able to forecast the behavior of the process is used in order to help the controller finding the optimal control scheme. These approaches successfully handle several inputs of dynamic non-linear systems. They are used in particular in the domain of car engines [6] but are limited by the construction of the model on one hand, and its parametrization on the other hand, which can take several years.

Intelligent Control Intelligent control can be seen as a part of adaptive control. It regroups control approaches that use Artificial Intelligence methods to enhance the controller adaptiveness. Among these methods we can find neural networks [10], fuzzy logic [12], expert systems [18] and bayesian controllers [5]. Neural networks provide

great genericity but they require an offline learning step with a considerable amount of data. Even if this enable an efficient control, they remain sensible to perturbations and cannot adjust themselves to the evolution of the system.

Expert systems are composed of a knowledge base and an inference engine. They lack of genericity and are hardly adaptable to highly dynamic non-linear systems. That is why they are often combined with fuzzy logic. Bayesian controllers are based on bayesian probabilities, the most common are the Kaman filter based controllers [13]. Yet, the bayesian approach does not deal with the problem of control itself but with a sub-problem which is the observation of the process. These methods can also be easily combined one with another, or with more classical control approaches.

Finally agents-based techniques have been applied in various fields, for example in the case of voltage control [22] or manufacturing control [4].

2.2 Self-Organizing Software

The next paragraphs introduce the notion of self-organization along with the Adaptive Multi-Agent Systems theory.

Self-Organization The notion of self-organization refers to the fact that the organization of a system results from internal mechanisms and local interactions without any explicit external control [3]. Various artificial systems show this property, such as swarms [14] or some multi-agent systems [16]. Self-organization brings robustness and scalability, and allows the engineering of underspecified software [7].

Adaptive Multi-Agent Systems The Adaptive Multi-Agent Systems (AMAS) theory is a basis for the design of multi-agent systems where cooperation is the engine for self-organization [8]. As cooperative entities, AMAS agents try to reach their own goals as well as they try to help other agents to achieve their goals. Moreover, an agent will modify its behavior if it thinks that its actions are useless or detrimental to its environment. Such situations are called Non-Cooperative Situations (NCS). Some behavioral rules are specific to NCSs, they allow agents to avoid or to solve these situations. By solving NCSs in regard to their own local goals, cooperative agents collectively find a solution to the global problem. Therefore one can consider the global behavior of an AMAS as emergent.

AMAS principles have been applied by [20] for the control of bioprocesses, avoiding the use of a specific model and dealing with the evolution of the process over time, since an AMAS is perpetually adjusting itself to its environment.

The next two sections describes the agents of our controller, how they perform control and how they self-organize.

3 CONTROLLER OVERVIEW

In this section we present the required abilities of a complex system controller, and what are the agents of our controller that enable them.

3.1 Basic Behavior

The next paragraphs describe how our multi-agent system, called ESCHER (Emergent Self-adaptive Controller for Heat Engine Regulation), works when it is already adapted to the engine. The mechanisms that lead to this adaptation will be explained further.

Observing the Process If we intend to control an engine, it is obvious that we need to be able to observe it. A specific agent type is in charge of perception, called *Variable Agent*. Each input and each output of the controlled black box has a corresponding Variable Agent. These agents perceive their value from the black box and send it to other agents that need this information. Also, Variable Agents can embed noise reduction algorithms if this problem is not handled by a third party system.

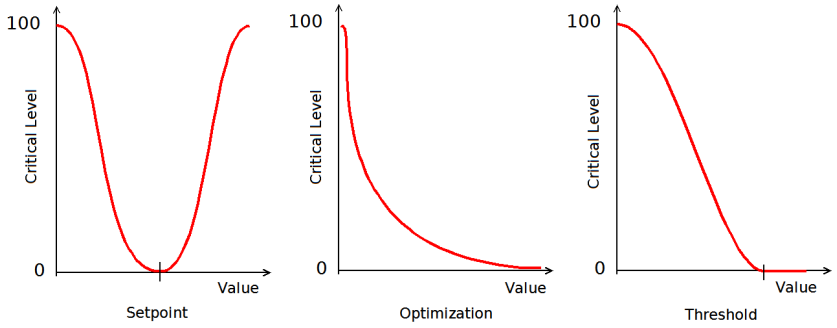

Fig. 1: Criticality Functions

Representing Objectives and Constraints The controller needs to know what is the desired state of the engine. This state is represented by a set of *Constraint Agents* and possibly by additional Variable Agents.

There are three types of Constraint Agents: Threshold, Setpoint and Optimization. A Threshold Constraint Agent expresses the will to keep a variable either below or above a threshold specified by a Variable Agent. This Variable Agent can either correspond to a process variable or to a user-defined value. In a similar way, a Setpoint Constraint Agent expresses the will to set a process variable to a particular value specified by a Variable Agent. Finally, an Optimization Constraint Agent represents the will to minimize or maximize a process variable.

Using the functions shown in figure 1, each Constraint Agent computes a *critical level* that varies from 0 (the agent is satisfied) to 100 (the agent is far from satisfied). The critical level depends on the value of the variable on which the constraint is applied and can be parametrized by a second Variable (in the case of a threshold or a setpoint constraint). The criticality function of an Optimization Constraint Agent is asymptotic to zero, since we always try to minimize or maximize the value. For instance, if the user desires to set the rotational speed to 5000rmp, the Variable Agent RotSpeedCommand

(corresponding to “user-defined setpoint on rotational speed”) will be set to 5000 and a setpoint Constraint Agent will be created. Its critical level will be zero if the value of the Variable Agent RotSpeed equals the value of RotSpeedCommand, and will increase with the difference between RotSpeed and RotSpeedCommand.

It is clear that decreasing the critical levels means solving the constraints, and the only way to do so is to perform the adequate actions on the engine inputs. Finding these actions implies to be able to analyze the current state of the environment. The engine and the user desires are considered as the environment of the multi-agent system.

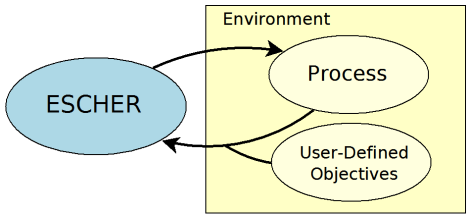

Fig. 2: ESCHER and its Environment

Analyzing the State of the Environment As shown in Figure 2, the environment of ESCHER is the engine as well as the user-defined setpoints, thresholds and optimizations. Thanks to Variable Agents and Constraints Agents, the ESCHER has a representation of its environment. Before it can perform control, it must be able to extract relevant information from this representation. This is the role of agents called *Context Agents*.

A Context Agent memorizes the effects, on every critical level, of an action applied to one particular input of the engine. It also memorizes the state of the environment when the action was applied. To represent this state the Context Agent maintains a set of *validity ranges*, containing one range per Variable Agent. A Context Agent also maintains a set of values that are the observed and memorized effects of the action on each of the critical levels. This set is called the *forecasts*. In other words, a Context Agent represents the information that *if every Variable Agent value is inside the Context Agent validity range, and if its action is applied, then the effects on every critical level will be similar to the Context Agent forecasts*.

A Context Agent is said *valid* when the environment is in a state that matches its validity ranges, i.e. every perceived variable is inside the corresponding validity.

When valid, a Context Agent sends a notification with its action and forecasts to the appropriate *Controller Agent*, which will be explained in the next part.

Selecting the Adequate Action Each controlled input of the engine is associated with a Controller Agent. The role of a Controller Agent is to apply the most adequate action in order to reduce the critical levels. It will base the choice of the action (ie increment, decrement or stay put) on the information it receives from Context Agents, picking the

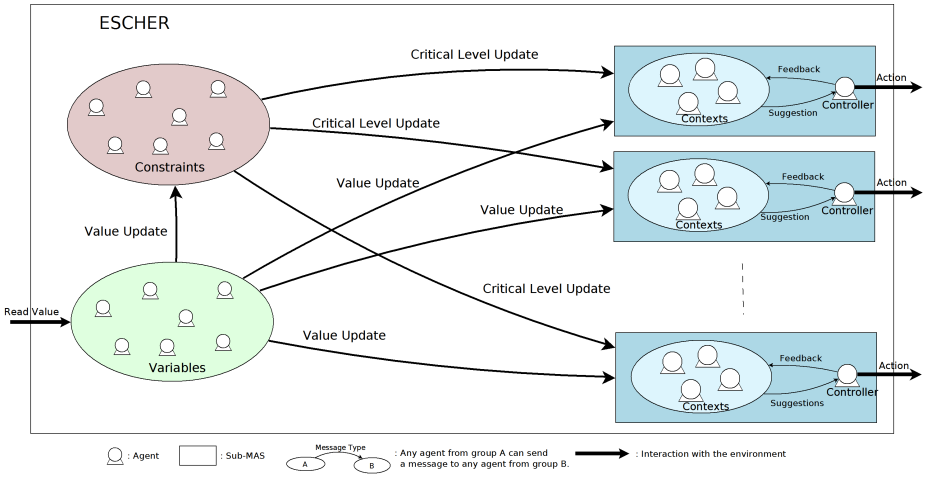

Fig. 3: ESCHER Topology

action that will provoke the biggest decrease of the critical levels. When an action is picked, the Controller Agent notifies every Context Agents who proposed it.

For instance, the Controller Agent of the mass fuel injected may receive two notifications from two Context Agents. Context Agent A says if you apply a decrease of 0.5 the critical levels will not change except for the Constraint Agent Torque Optimization that will rise of 3.5. Context Agent B says if you apply an increase of 0.5 the critical levels will not change except for the Constraint Agent Torque Optimization that will drop of 1.5. In this case, the Controller Agent obviously chooses to increase the mass fuel injected.

There are several cases where the Controller Agent is unable to make a good decision, because of incomplete or incorrect information from Context Agents. These cases are *Non-Cooperative Situations (NCS)*. They happen when ESCHER is not well adapted to the engine yet. When a NCS occurs, the cooperative behavior of involved agents is triggered in order to solve it. This will be explained in section 4. For now, we continue with the architecture of the system.

3.2 Topology

The four ESCHER's agent types were introduced in the previous paragraphs. Figure 3 shows the topology of their interactions. We can see that, while it is a MAS itself, ESCHER is also an aggregation of several MASs, each constituted by a Controller Agent and its set of Context Agents. The environment of these sub-MASs is composed of the Variable Agents, the Constraint Agents, and the environment of ESCHER. Indeed, each Controller Agent is not directly in interaction with other Controller Agents. The only link between them are their environment (the engine they are controlling, via the Variable and Constraint Agents). Actions from one Controller Agent will have consequences on the engine that will be reflected by Variable and Constraint Agents, and

Controller Agents share the same environment, hence we can expect them to progressively synchronize their actions. Furthermore, Context Agents never directly interact with each other (even inside the same set) but only with their associated Controller Agent.

The question is now how can the system create and adjust Context Agents in order to deal with an unknown process? The next section gives an answer.

4 CONTEXT AGENTS LEARNING AND ORGANIZATION

This section explains how the set of Contexts Agents is built and how they self-organize to provide an accurate representation of the explored part of the process' state space.

4.1 Non-Cooperative Situations

In this part, we explain how agents solve the Non-Cooperative Situations they face. Since they provoke changes in the agents' behavior, NCSs are the key of the adaptiveness of Adaptive Multi-Agent Systems. Each agent locally solves its own NCSs when they occur, by changing its own local behavior, and thus of the global behavior of the system. The NCSs of ESCHER concern the Controller Agents and the Context Agents, and trigger behaviors that correct Context Agents' forecasts and validity ranges.

No Adequate Action in Suggestions This NCS occurs when the suggestions list of a Controller Agent contains only forecasts of increasing critical levels. Whatever the Controller Agent may chose, it believes it will deteriorate the constraints. There are two cases: either all the possible actions are already suggested (i.e. increase the value, decreasethe value and stay put) or some actions are not suggested. In the first case, the only choice left is to accept the suggestion with the less bad forecasts. In the second case, a new action is chosen among those which are not suggested, and a new Context is created with this action.

Empty Suggestions List This NCS happens when a Controller Agent has to apply an action, but finds its suggestion list empty. It will be unable to find an adequate action with certainty, but it can make some hypothesis to try one. If the last applied action had reduced the higher critical level, the same action is reproduced. If not, the opposite action is applied (e.g. if the last action was an inscrease of the input value, the new action will be a decrease). If no action has ever been applied, the solution is to randomly chose the action. In any case a new Context Agent is created with the applied action. After its creation, a Context Agent extends its validity ranges as long as its action is applied and observe the variations of the critical levels to set its forecasts.

Wrong Forecast When a Context Agent is selected and its action applied, it oberves the variations of the critical levels to check if they matches its forecasts. If the forecasts are wrong (i.e. a critical level evolves in the opposite direction of the forecast), the Context Agent considers that it should not have been valid when the action was suggested. Thus

it reduces its validity ranges but does not modify its forecasts. A Context Agent dies if one of its ranges is reduced to an amplitude of zero. If the forecasts are simply inexact (i.e all the critical levels evolve in the forecasted direction, but not with the forecasted amplitude), the Context Agent considers that its validity was relevant. Thus it does not modify its validity ranges but adjusts its forecasts to match its observation.

This NCS also impacts the Controller Agent that had selected the Context Agent with the wrong forecasts. When a Controller Agent notices that the critical levels variations don't match the forecast, it stop applying the current action, notifies the Context Agent that its action is no longer applied, and looks for a new action in its suggestion list.

4.2 Self-Organization

Context Agents are able to evaluate their own behavior and to adjust it (by modifying their forecasts and their validity ranges) if necessary. Thus we can say that a Context agent is self-adaptive. Moreover, Context Agents are created at runtime. Each of them follows simple and local behavioral rules. These rules lead to the formation of a coherent set of Context Agents where each agent occupies a portion of the environment state space for which its forecasts on critical levels are correct. Thus we can say that the set is the result of the self-organization of the Context Agents. It is also intersting to note that all the Context Agents of a given set put together give a good approximation of the criticality functions.

There is another, more subtle, level of self-organization. Each Controller Agent makes its own decisions about the action to apply on its effector, other Controller Agents are never consulted. Nonetheless, it appears that they manage to synchronize their actions and efficiently reduce the critical levels, solving the constraints. This is possible because each process input corresponding to a Controller Agent is also represented as a Variable Agent. Context Agents suggestions for one Controller Agent are therefore conditionned by the current state of other Controller Agents corresponding process input. As they all try to lessen the critical levels, they eventually find a synergy and the global behavior of ESCHER (the actions on all the controlled inputs) is coherent.

5 RESULTS

Generated black-boxes [2] were used for the first tests to check ESCHER's abilities to deal with multiple inputs and outputs, to find compromises between several objectives, and to adapt to unexpected changes of behavior of the controlled process. These tests are not detailed here due to space limitations.

Several tests have been driven on a 125cc monocylinder fuel engine in various steady operating points of rotational speed and load. Those presented here have been driven at 5000rpm and 870mbar of pressure in the inlet manifold. The goal of ESCHER is here to maximize the indicated mean effective pressure (IMEP), an output variable that reflects the torque. In other words, the goal is to minimize the critical level of an Optimization Constraint Agent associated to the IMEP. Injected mass fuel (IMF) and ignition advance are controlled by ESCHER, which has no other previous information

about the engine than the variation interval of the variables. Each test has been supervised and validated by domain experts.

5.1 Controlling Mass Fuel Injection

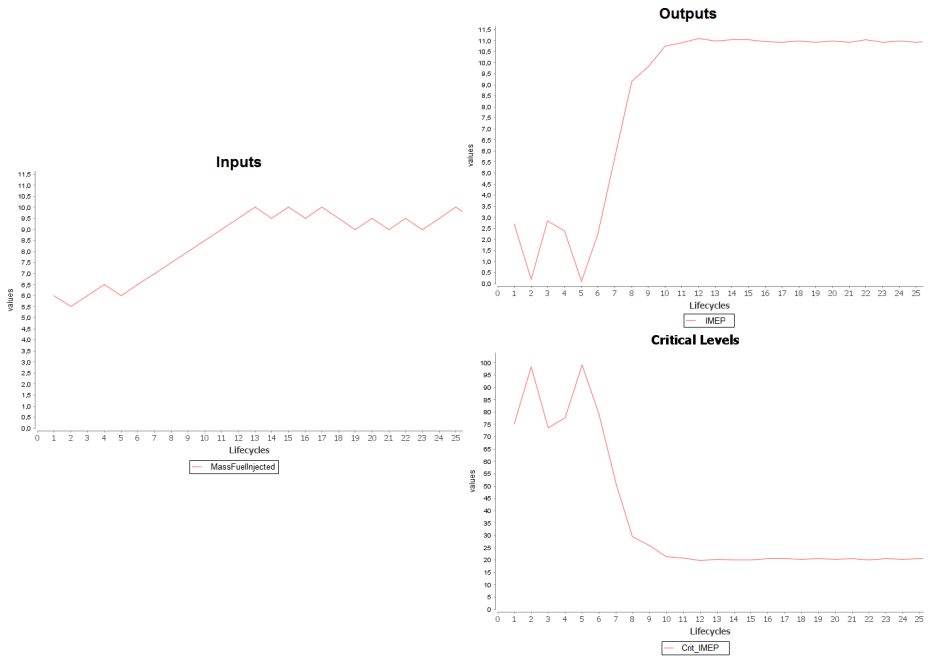

Fig. 4: Controlling Injected Mass Fuel to Maximize IMEP

Figure 4 shows ESCHER controlling the IMF to maximize the IMEP. At the beginning of the test, ESCHER does not know anything about the controlled process. It first decreases the IMF, but it is a mistake: the critical level rises (first lifecyle). The error is corrected by increasing the IMF. The critical level drops, so ESCHER keeps this action (lifecycles 2 and 3). Then it perceives that while the IMF is rising, the critical level of the IMEP is rising too. But it is due to the noise on the IMEP. This causes ESCHER to mistakenly decrease the IMF. The error is rapidly detected and ESCHER eventually finds that it has to increase the IMF in order to reduce the critical level (lifecycles 5 to 13). Once the maximal IMEP has been reached, ESCHER oscillates around the IMF value that keeps the critical level at its minimum. During this test, five Contexts Agents were created and the optimum value was reached in 12 lifecycles.

5.2 Controlling Mass Fuel Injection and Ignition Advance

In this test, ESCHER controls the ignition advance in addition to the IMF. Figure 5 shows the curves of this test. Results are similar to the previous test, with ESCHER

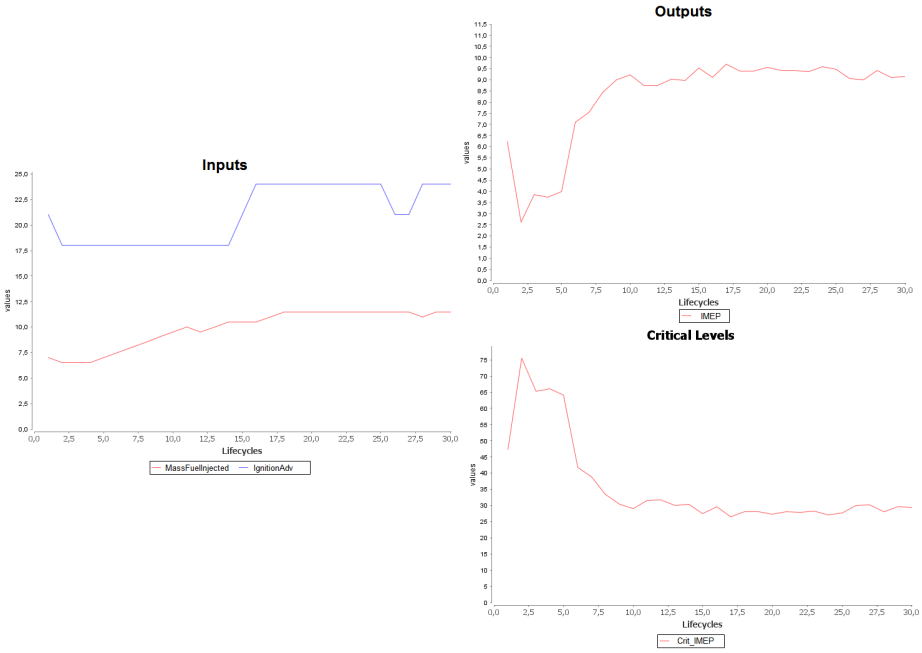

Fig. 5: Controlling Injected Mass Fuel and Ignition Advance to Maximize IMEP

first committing a mistake on both of the controlled inputs, causing a rise of the critical level. Then the error is corrected, before the controller stabilizes the engine around the maximum IMEP.

The optimum IMEP was reached in 18 lifecycles. During the 30 lifecycles of the test, ESCHER created 19 Context Agents (10 were created by Mass Fuel Injected Controller Agent and 9 by Ignition Advance Controller Agent).

The main challenge with the real engine compared to generated black boxes is to deal with noise and latency. As ESCHER quickly reacts to errors, the noise is handled naturally. To overcome the latency between an action and its effects, ESCHER waits during a parametrizable period of time between its actions and its observations. For each test on the engine, a lifecycle was executed every three seconds.

5.3 Discussion

ESCHER finds optimum values for all of the tested operating points. As it is impacted by the noise on the variables, the time taken to converge may vary. The biggest differences were observed at 4000rpm and 700mbar of pressure in the inlet manifold where it took from 100 seconds up to 400 seconds to converge. But it remains quicker than human experts doing it by hand as it is currently the case in the industry. More tests are planned, with more objectives and more simultaneously controlled variables.

6 CONCLUSION AND FUTURE WORKS

In this paper, we presented an Adaptive Multi-Agent System, called ESCHER, which controls heat engines, basing its behavior on self-organizing agents. We detailed its architecture and the agents interactions.

ESCHER is based on the idea that the complete understanding of how an engine works is not necessary to its control. All the controller needs to know is how the engine behaves. In other words our controller is of black-box type: ESCHER only perceives the engine's inputs and outputs, but not its internal mechanisms. The basic principle of our controller is the following: the state of the engine inputs/outputs is memorized when an action is applied and the consequences of this action are recorded. This information will be used to decide whether this action was good or not in regard of the user-defined desired engine state. This means that the quality of the control improves over time: at the begining the controller knows nothing about the engine, but it perpetually learns from its actions and quickly manages to control it. The learning is parallel to the control, so ESCHER continuously self-adapts to the process.

No prerequisite knowledge other than the intentions of the user (i.e. some criticality functions) is needed by ESCHER. It is also able to satisfy multi-criteria constraints on multiple inputs and outputs. Moreover, the independence between Controller Agents gives a certain modularity to ESCHER. Each Controller Agent (and its related Context Agents) is a stand-alone MAS that can be plugged on a process input. Regardless on what is controlling the other inputs, it will be able to synchronize its actions to perform a correct control.

We showed that ESCHER is able to deal with non-linearity and multiple inputs and outputs, and that it handles the noise and latency of a real engine, without heavy knowledge about the engine. The only required instantiation work is the translation of the user's objectives into criticality functions.

By focusing on its learning skills, we consider to make the ESCHER generic enough to be easily applied to all kind of systems. Multi-Agent Systems using a similar pattern have already been successfully applied to the control of the temperature of a bioprocess [20] and are currently being applied in the contexts of ambiant systems [9] and energy management of buildings.

Secondly, the behavior itself needs some improvements. New mechanisms for a better resolution of the problem of latency between the actions and their effects are required. Moreover, in large processes, some inputs may not affect some outputs. The learning of the process behavior could be enhanced by taking into account this fact. Finally, the main limitation to the application of ESCHER is the need for predefined criticality functions. Self-defined criticality functions could simplify make our controller even easier to instantiate.

References

1. Karl Johan Astrom and Tore Hagglund. *PID Controllers: Theory, Design, and Tuning*. Instrument Society of America, Research Triangle Park, NC, second edition, 1995.
2. Jérémy Boes, Pierre Glize, and Frédéric Migeon. Mimicking Complexity: Automatic Generation of Models for the Development of Self-Adaptive Systems. In *International Conference*

on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), Reykjavik. INSTICC Press, july 2013.
3. Scott Camazine, Jean-Louis Deneubourg, Nigel R Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. *Self-Organization in Biological Systems.* Princeton University Pres, 2003.
4. G. Clair, E. Kaddoum, M.-P. Gleizes, and G. Picard. Self-regulation in self-organising multi-agent systems for adaptive and intelligent manufacturing control. In *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2008.
5. Bianca M. Colosimo and Enrique Del Castillo, editors. *Bayesian Process Monitoring, Control and Optimization*. Taylor and Francis, Hoboken, NJ, 2007.
6. Marcelin Dabo, Nicolas Langlois, Witold Respondek, and Houcine Chafouk. NCGPC with dynamic extension applied to a Turbocharged Diesel Engine. In *Proceedings of the International Federation of Automatic Control 17th World Congress*, pages 12065–12070, 2008.
7. Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, editors. *Self-organising Software - From Natural to Artificial Adaptation*. Natural Computing Series. Springer, octobre 2011.
8. Jean-Pierre Georgé, Marie-Pierre Gleizes, and Valrie Camps. Cooperation. In Giovanna Di Marzo Serugendo, editor, *Self-organising Software*, Natural Computing Series, pages 7–32. Springer Berlin Heidelberg, 2011.
9. Valérian Guivarch, Valérie Camps, and André Péninou. Context awareness and adaptation in ambient systems by an adaptive multi-agent approach. In *International Joint Conference on Ambient Intelligence, Italy*, 2012.
10. Martin T. Hagan, Howard B. Demuth, and Orlando De Jesus. An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control*, 12(11):959–985, 2002.
11. Gerhard Kreisselmeier and Brian Anderson. Robust model reference adaptive control. *IEEE Transactions on Automatic Control*, 31(2):127 – 133, feb 1986.
12. Chuen Chien Lee. Fuzzy logic in control systems: Fuzzy logic controller. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418, mar/apr 1990.
13. Jay H. Lee and N. Lawrence Ricker. Extended kalman filter based nonlinear model predictive control. In *American Control Conference*, pages 1895–1899, june 1993.
14. Alberto Montresor, Hein Meling, and zalp Babaolu. Messor: Load-balancing through a swarm of autonomous agents. In Gianluca Moro and Manolis Koubarakis, editors, *Agents and Peer-to-Peer Computing*, volume 2530 of *Lecture Notes in Computer Science*, pages 125–137. Springer Berlin Heidelberg, 2003.
15. M. Nikolaou. Model predictive controllers: A critical synthesis of theory and industrial needs. *Advances in Chemical Engineering*, 26:131–204, 2001.
16. Greg MP O'Hare and Nicholas R Jennings. *Foundations of distributed artificial intelligence*. Wiley-Interscience, 1996.
17. Torsten Soderstrom and Petre Stoica. *System identification*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
18. Robert F. Stengel. Intelligent failure-tolerant control. *IEEE Control Systems*, 11(4):14–23, june 1991.
19. Bjorn D. Tyreus and William L. Luyben. Tuning PI controllers for integrator/dead time processes. *Industrial & Engineering Chemistry Research*, 31(11), 1992.
20. Sylvain Videau, Carole Bernon, Pierre Glize, and Jean-Louis Uribelarrea. Controlling Bio-processes using Cooperative Self-organizing Agents. In Yves Demazeau, editor, *PAAMS*, volume 88 of *Advances in Intelligent and Soft Computing*, pages 141–150. Springer-Verlag, 2011.
21. Antonio Visioli. Tuning of PID controllers with fuzzy logic. *IEE Proceedings - Control Theory and Applications*, 148(1):1–8, jan 2001.

22. H.F. Wang. Multi-agent co-ordination for the secondary voltage control in power-system contingencies. *Generation, Transmission and Distribution, IEEE Proceedings*, 148(1):61 –66, jan 2001.