Model-free Optimization of an Engine Control Unit thanks to Self-Adaptive Multi-Agent Systems

Jérémy Boes[1], Frédéric Migeon[1], Pierre Glize[1], Erwan Salvy[2]
[1]*Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier, Toulouse, France*
[2]*Aboard Engineering, Toulouse, France*
{*boes, migeon, glize*}*@irit.fr, erwan.salvy@aboard-eng.com*

Keywords: Control ; Multi-Agent Systems ; Self-Organization ; Auto-Calibration ; Automotive

Abstract: Controlling complex systems, such as combustion engines, imposes to deal with high dynamics, non-linearity and multiple interdependencies. To handle these difficulties we can either build analytic models of the process to control, or enable the controller to learn how the process behaves. Tuning an engine control unit (ECU) is a complex task that demands several months of work. It requires a lot of tests, as the optimization problem is non-linear. Efforts are made by researchers and engineers to improve the development methods, and find quicker ways to perform the calibration. Adaptive Multi-Agent Systems (AMAS) are able to learn and 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. In this paper, we describe a multi-agent control system that was used to perform the automatic calibration of an ECU. Indeed, the problem of calibration is very similar to the one of control: finding the adequate values for a system to perform optimally.

1 INTRODUCTION

Before selling a vehicle, manufacturers must seek registration, in other words, they need to check their compliance with respect to different regulations. Vehicles are subject to all sorts of regulations regarding active and passive safety, the environment, manufacturing, and so on.

For instance, in order to improve air quality in Europe, gaz emissions are limited by law. Limits were lowered several times in recent decades, gradually passing from Euro 1993 to Euro V standard that is in force today. The latest European air regulations require manufacturers to change their vehicles, sometimes in depth, and develop combustion engines with more and more complex technical definitions, causing many changes on the electronic architecture. Systems like EGR valves, variable displacement oil pumps, electric water pumps, variable geometry turbo, are some of the components to be controlled by the electronic control unit, the ECU (also stands for engine control unit).

All of these elements are controlled by the ECU via functions called control strategies. The calibration engineers must tune this increasing number of functions to meet the standards. This greatly increases the calibration and test workload, and force to apply complex development methodologies. In this context, manufacturers and suppliers must constantly innovate to produce new engines while reducing the time and cost of development. New approaches such as design of experiments or automatic calibration are becoming essential to keep the competitiveness.

The automatic calibration tools available on the market are integrated to engine dyno management software solutions. To perform a test sequence, technicians and engineers define the number and area scan settings, the constraints and output to optimize, then proceed to the more conventional setting of the bench (acquisition channels, security, appliances measurement, etc). When a test is started, the control software of the engine dyno is autonomous, it no longer requires human intervention to go to the end of the sequence. Once the sequence is completed, engineers analyze the acquired data, and select the optimum settings that meet their specifications. This test sequence is repeated as many times as there are operating points to optimize.

By comparison, ESCHER, the multi-agent system presented in this paper, applied to the development of engine tuning, works differently than existing tools. The essential differences compared to other tools are

that it is separated from the control software of the engine dyno, and that it offers optimum setting within some minutes. ESCHER requires very few configuration: engineers select only the constraints to be met, and parameters to be optimized, then the tool can start optimizing the current operating point. This way, the long tests sequences involved in the usual full scan of the parameters, as well as the heavy processing of the huge amount of acquired data, are avoided. It is of course necessary to set the bench on its conventional aspects. ESCHER performs optimization within some minutes depending on the number of parameters to modify, and the number of responses to be optimized.

In section 2, we briefly present various control methods and approaches that have been experimented and used over the years. We follow with an introduction to the Adaptive Multi-Agent Systems theory. In section 3 the design of our controller is described before going a bit deeper in the agents behavior. Section 4 gives details about the implementation of our system, while section 5 exposes the light effort required for the application of our approach in real cases. Section 6 presents the experiments and the results obtained with a real engine. Finally, section 7 concludes with our perspectives and future works.

2 RELATED WORKS

In this section the main approaches of control are presented before a brief introduction to the Adaptive Multi-Agent Systems theory.

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 most well-known are presented in the next paragraphs.

PID - The widely used Proportional-Integral-Derivative (PID) controller computes three terms related to the error between the current and the desired state of the process, from which it deduces the next action to apply (Astrom and Hagglund, 1995). PID controllers are not efficient with complex systems, due to their difficulties to handle several inputs and outputs and to deal with non-linearity.

Adaptive Control - Model-based approaches like Model Predictive Control (MPC) (Nikolaou, 2001) use a model able to forecast the behavior of the process in order to find the optimal control scheme. These approaches handle several inputs but are limited by the mathematical models they use. The Dual Control Theory uses two types of commands : the actual controls that drive the process to the desired state, and probes to observe the process reactions and refine the controller's knowledge (Feldbaum, 1961). The concept of this approach is interesting but a heavy instantiation work is still required.

Intelligent Control - Intelligent control regroups approaches that use Artificial Intelligence methods to enhance existing controllers. Among these methods we can find neural networks (Hagan et al., 2002), fuzzy logic (Lee, 1990), expert systems (Stengel, 1991) and bayesian controllers (Colosimo and Del Castillo, 2007). These methods can be easily combined one with another.

Engine Control and Calibration Most engine control methods use mathematical models, such as the Jankovic third-order model (Jankovic and Kolmanovsky, 2000). The challenge is to deal with the non-linearity of the model to compute the best actions on the engine actuators. For instance, (Dabo et al., 2008) use dynamic feedback linearization to handle the exhaust gas recirculation valve of a turbocharged diesel engine.

Automatic calibration methods also rely on models of the engine. The most advanced techniques allow the controller to learn the best value of a part of its own parameters. For instance, (Malikopoulos et al., 2009) use a stochastic approach to find the calibration of a diesel engine. They model the engine and the optimization criteria as a Markov Decision Process, and use a decentralized reinforcement learning algorithm to find the best injection timing and VGT postion.

2.2 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 (Georgé et al., 2011). 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, specific to NCS's, help agents to solve or avoid these situations. By solving NCS's, in regard to their own local goals, cooperative agents collectively find a solution to the global problem. Therefore one can consider the behavior of an AMAS as emergent.

Thanks to its adaptiveness, an AMAS-based controller should not rely on a specific model of the pro-

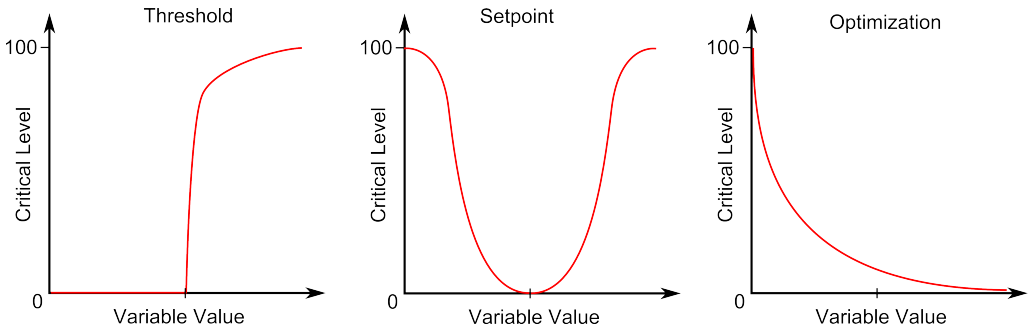

Figure 1: Examples of criticality functions

cess thus it does not need a heavy instantiation work. Besides, it should be able to deal with a changing number of inputs and outputs.

3 CONTROLLER OVERVIEW

Controlling a system basically means finding the most adequate action to apply on its inputs in order to obtain the desired effect on its outputs. Here we present the required basic abilities of a complex system controller, and what are the agents that enable them. Then, we explain with a simple example how it is possible to control a process with local behavioral rules.

3.1 Nominal Behavior

The next paragraphs describe how our multi-agent system, called ESCHER (for Emergent Self-Adaptive Control for Heat Engine calibRation), works when it is already adapted to the controlled system. The mechanisms leading to this adaptation will be explained further.

3.1.1 Observing the Process

If we intend to control a system, it is obvious that we need to be able to observe it. A specific agent type is in charge of perception, called *Variable Agent* (there is one for each input and output of the controlled process). These agents perceive their value from the process and send it to agents who need this information. Also, Variable Agents can embed noise reduction algorithms if this problem is not handled by a third party system.

3.1.2 Representing Criteria

The controller needs to know what is the desired state of the process. This state is represented by a set of *Criterion Agents* and possibly by additional Variable Agents.

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

Each Criterion 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 Criterion is applied and can be parametrized by a second Variable (in the case of a threshold or a setpoint criterion). Figure 1 shows examples of *criticality functions* that can be used to compute the critical level of a threshold, a setpoint, and an optimization criterion.

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 process inputs. Finding these actions implies to be able to analyze the current sate of the environment.

3.1.3 Analyzing the State of the Environment

ESCHER's environment is the process to control as well as the user-defined setpoints and thresholds. Thanks to Variable Agents and Constraints Agents, the multi-agent system 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 and expresses the effects, on every critical level, of an action applied to one particular input of the process. 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

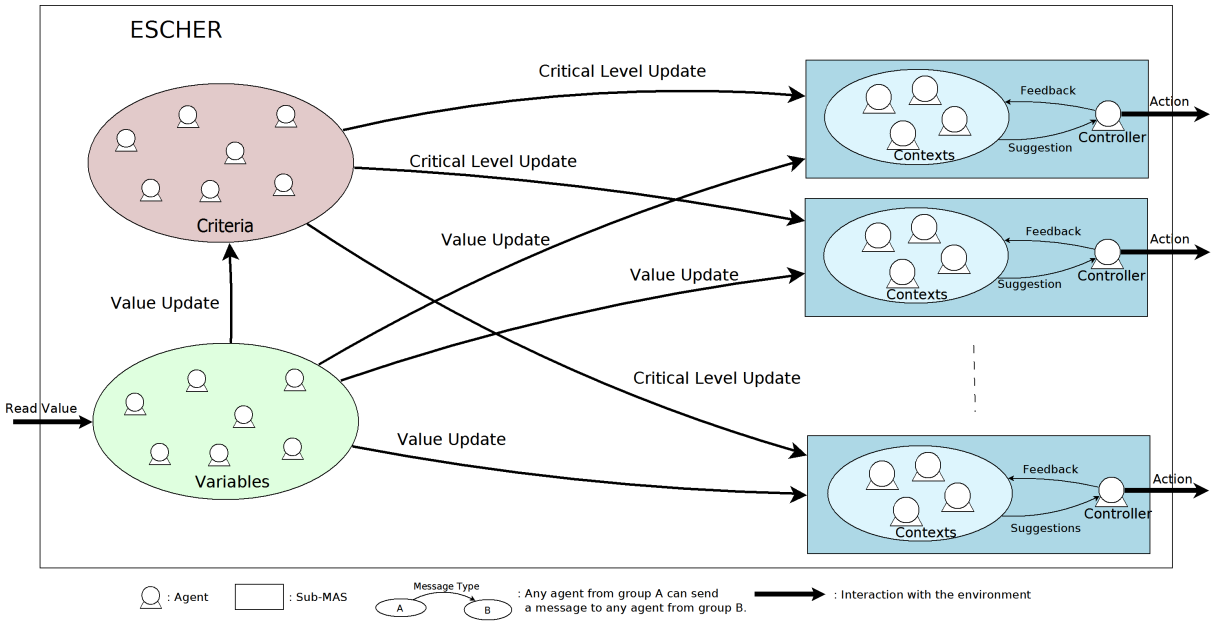

Figure 2: ESCHER Topology

per Variable Agent. The memorized effects on critical levels form its *forecasts*. In other words, a Context Agent represents the information that *if every variable value is inside its validity range, and if this action is applied, then the effects on every critical level will be similar to these forecasts*.

A Context Agent is said *valid* when the environment is in a state that matches its validity ranges. When this occurs, it sends a notification with its action and forecasts to the appropriate *Controller Agent*, which will be presented in the next part.

3.1.4 Selecting the Adequate Action

Each controlled input of the process 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 its choice on the information it receives from Context Agents, picking the action that will provoke the biggest descrease of the critical levels. When an action is picked, the Controller Agent notifies every Context Agents who proposed it.

Figure 2 shows the global architecture of the system. 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)*. When a NCS occurs, the cooperative behavior of involved agents is triggered in order to solve it. This will be explained in 3.2.

3.2 Non-Cooperative Situations

This section presents the main NCSs that our agents face and how they solve it, leading the system to have an accurate representation of the process to control.

No Adequate Action in Suggestions - This NCS occurs when the suggestions list of a Controller Agent contains only forecasts of increasing critical levels. There are two cases : either all the possible actions are already suggested or some actions are not proposed. 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 applied, 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 action it applied had reduced the maximum of the Criterion Agents critical levels, the same action is reproduced. If not, the opposite action is applied. A new Context Agent is created, with the applied action. After its creation, a Context Agent will extend its validity ranges as long as its action is applied.

Wrong Forecast - This NCS occurs when a Context Agent is selected, checks its forecasts and notices that they are not correct. 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 : its validity ranges are reduced. A Context

Agent dies if one of its ranges is reduced to an amplitude of zero. If the forecasts are only erroneous (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 does not modify its validity ranges, but adjusts its forecasts to match its observation.

3.3 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.

4 IMPLEMENTATION

This section gives some details about the implementation of ESCHER.

4.1 Component-based Architecture

ESCHER has a component-based architecture. It has been implemented using a tool called MAY (Make Agents Yourself). MAY is an Eclipse plug-in that allows to describe the architecture of the agents and the infrastructure that support them. Based on this description, a JAVA code skeleton is automatically generated (Noël, 2012). In ESCHER, the four types of agents have the same architecture, and only differ by the implementation of each component. Figure 3

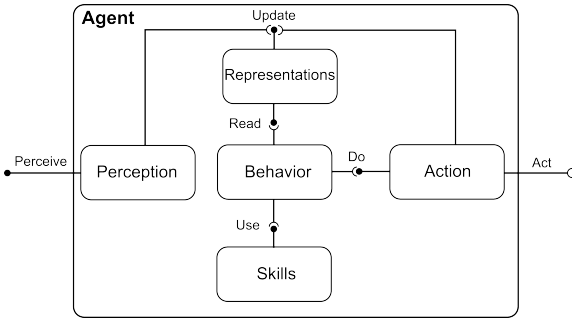

Figure 3: Component-based architecture of an agent.

shows the main components of the agents:

- Perception: contains the methods to receive messages, and extract relevant informations. For instance, the perception component of Variable Agents is used to read messages as well as the variable value on the process.
- Representations: stores all the data extracted by the perceptions component, and domain specific knowledge of the agent. For instance, the validity ranges of a Context Agent are located in its representations component.
- Skills: contains useful methods to help the agent in its decisions.
- Behavior: contains the behavior rules of the agent. These ruled produce actions in regard to the analysis of the current state of the representations.
- Action: contains the methods to execute actions. Actions can be sending messages, tuning internal parameters, or (only for Controller Agents) setting the value of a parameter of the process.

These components can be made of other generic components, that can be reused thanks to MAY (a good example is the inbox for the message passing).

4.2 Adaptive Value Trackers

We have seen in section 3.2 that Context Agents tune their internal parameters themselves (like their forecats, or their validity ranges). This tuning is done with Adaptive Value Trackers (AVT). AVTs use simple feedbacks to converge towards a value (Lemouzy et al., 2011). These feedbakcs are “greater” (the AVT has to increase its value), or “lower” (the AVT has to decrease its value). The amplitude of the variation

(later called Δ) is determined by the sequence of received feedbacks. If two consecutive feedbacks are identical, Δ is increased, otherwise, Δ is decreased.

An AVT is able to converge quickly towards a value, hold steady, and then converging towards a new value if necessary. Thus, they are adequate in our case, where the parameters of an agent (such as the forecats of a Context Agents) change over time.

5 APPLICATION

While designed as a control system, ESCHER can also be seen as an automatic calibration tool. Indeed, learning how to control an unknown engine is very similar to learning the optimal parameters of an ECU. This section explains how ESCHER is used during the calibration process, and what are its parameters.

5.1 Calibration Process

The classical calibration process of an ECU involves an early phase of mapping, where interesting operating points are chosen. This phase is usually followed by the complete scan of the calibration parameters (for each selected operating point), in order to collect data on the engine, which is later processed to find the optimal parameters. This step demands a lot of tests, since the combinatorial space of the parameters is huge, and finding the optimum is a complex task. This is why ESCHER is used. Instead of performing a full scan, we let ESCHER control all the parameters at once. It will explore the engine state space and converge towards the optimal configuration for the controlled parameters.

5.2 Parameters of ESCHER

ESCHER is easy to instantiate to a given engine. This means that no heavy parameter tuning is required, and that no model of the engine is needed. There are two types of parameters that must be set: the parameters relative to the engine, and the parameters relative to the optimization criteria. The only engine-related mandatory parameters are:

- the number of controlled variables, and the reference of each ;
- the number of observable variables, and the reference of each.

These parameters are used to create the adequate Variable Agents and Controller Agents. It is also possible to set the variation range of each variable. This is optional, and only useful if you want to restrain the exploration of the controlled variables (for instance, for safety purposes). Setting variation ranges also helps to set the parameters for the optimization criteria. Anyway, this is basic knowledge about the considered engine. Thus, there is no need to run extensive engine tests, neither to build a model of the engine, prior to apply our method.

We have seen, in section 3.1.2, that optimization criteria are represented by Criterion Agents. Each of these agents computes a critical level, thanks to a criticality function. These functions are defined by the user. Having the variation range of the variable on which a criterion is applied helps to get cost-efficient functions (otherwise, the use of the exponential function is required). A default polynomial function for each type of criterion is proposed, and implemented in ESCHER. With these functions, the user only have to select the type of criterion (setpoint, threshold, or optimization), and to specify a value (for a setpoint, or a threshold) and a direction (for a thershold, or an optimization).

Parameters	Significance
Number of controlled variables	Mandatory
Number of observable variables	Mandatory
Variables references	Mandatory
Variables variation ranges	Optional
Criticality functions	Mandatory
Maximal action	Not significant
Minimal size of the validity ranges	Not significant
Minimal Δ for AVTs	Not significant
AVTs coefficient	Not significant

Table 1: Parameters of ESCHER.

Finally, a few secondary parameters can be tuned, but do not significantly impact the behavior of ESCHER. These are the maximal amplitude of an action, the minimal size of a validity range, the minimal Δ for an AVT, and the coefficient for the self-tuning of this same Δ. Table 1 summarizes the parameters of ESCHER.

6 EXPERIMENTS

This section shows results obtained during live experiments on a real engine. The results presented in this paper have been obtained on a monocylinder 125cc fuel engine.

6.1 Installation

The ECU is accessed through a specific software, called ControlDesk. ESCHER and ControlDesk run

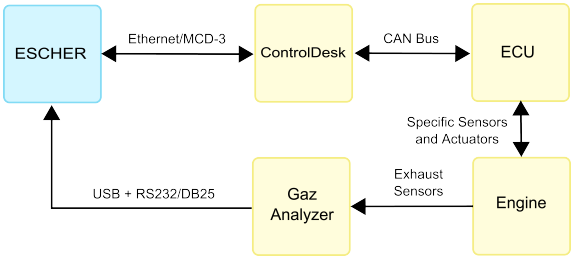

Figure 4: The experimental setting.

on different computers, and communicate through the MCD-3 protocol. This allows ESCHER to read and write any value on the ECU. The mass of injected fuel, the ingnition advance, and the start of injection, are all bypassed on the ECU, so ESCHER can directly control them. Finally, a gaz analyzer is plugged to the exhaust system and sends its data to ESCHER via its serial port. Figure 4 shows the communication means between the different systems.

6.2 Torque Optimization

In this test, the operating point is 5000rpm and 870mbar of pressure in the inlet manifold. The goal is to maximize the indicated mean effective pressure (IMEP), while controlling the injected mass of fuel (IMF) and the ignition advance. Figure 5 shows the curves of these three variables. The Variable Agent of the IMEP is therefore associated with an Optimization Criterion Agent, whose critical level is also shown in figure 5.

At the beginning of the test, ESCHER does not know anything about the controlled process (there is no Context Agent), it will learn from its actions and observations. At first, ESCHER decreases both of the controlled inputs. It is a mistake, as it leads to a rise of the critical level (since the IMEP decreases). ESCHER reacts by increasing the IMF at first, and the ignition advance a moment later, before finally stabilizing the values when the IMEP stops increasing. At the end of the test, the IMEP has been maximized.

During the 30 lifecycles of the test, ESCHER created 19 Context Agents (10 were created by the IMF Controller Agent and 9 by the Ignition Advance Controller Agent). The test lasted a total of 90 seconds.

6.3 Torque and Fuel Consumption Optimization with Pollution Thresholds

In this test, ESCHER controls the IMEP, the ignition advance, and the start of injection (SOI). The goal is to maximize the IMEP, while minimizing the specific

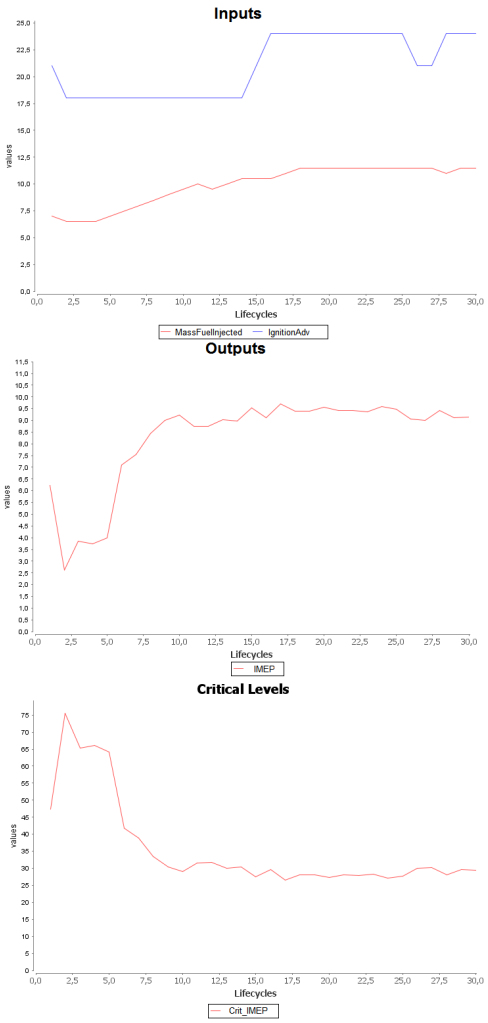

Figure 5: Torque Optimization with Injected Mass of Fuel and Ignition Advance

fuel consumption (SFC) and observing a threshold on hydrocarbons (HC, below 500ppm) carbon monoxide (CO, below 3%). The operating point for this test is 2500rpm and 750mbar in the inlet manifold. Figure 6 shows the manipulated parameters and the critical levels, while figure 7 shows the outputs of the engine.

At the beginning of the test, we see on figure 6 that the highest critical level is the one associated with the fuel consumption. Hence, ESCHER tries to reduce this one in priority. At first, ESCHER increases the IMF and the ignition advance, and decreases the SOI. This effectively reduces the SFC (and also increases the IMEP). But, as a side effect, there is a rise of gas emissions, as CO eventually crosses its threshold . At this point (around the lifecycle 20), its criterion becomes the most critical. The goal for ESCHER is then to lower it, without the other critical levels becoming higher. After several oscillations, ESCHER

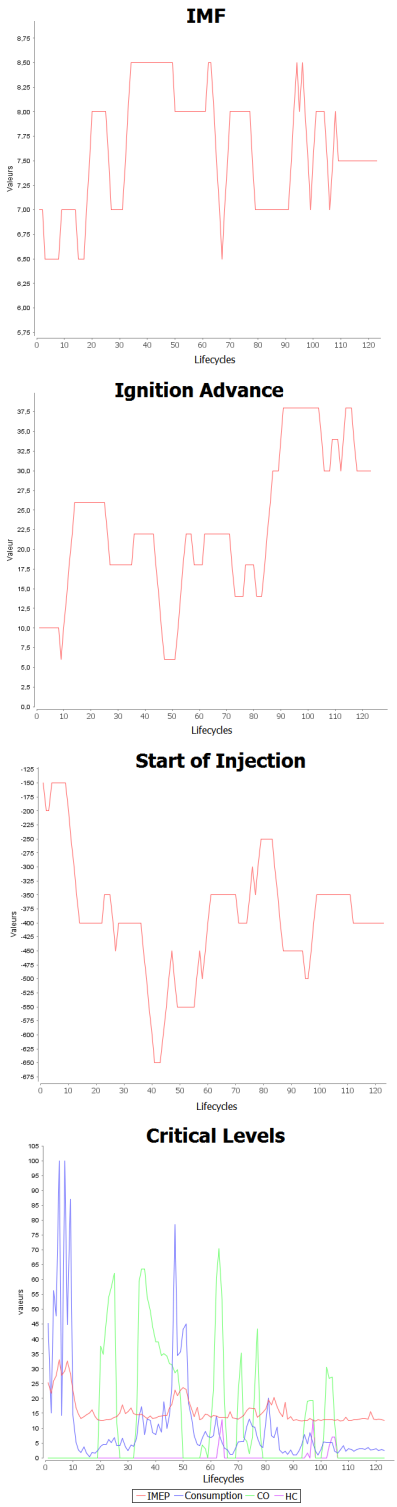

Figure 6: Torque and Fuel Consumption Optimization with Pollution Thresholds (inputs and critical levels)

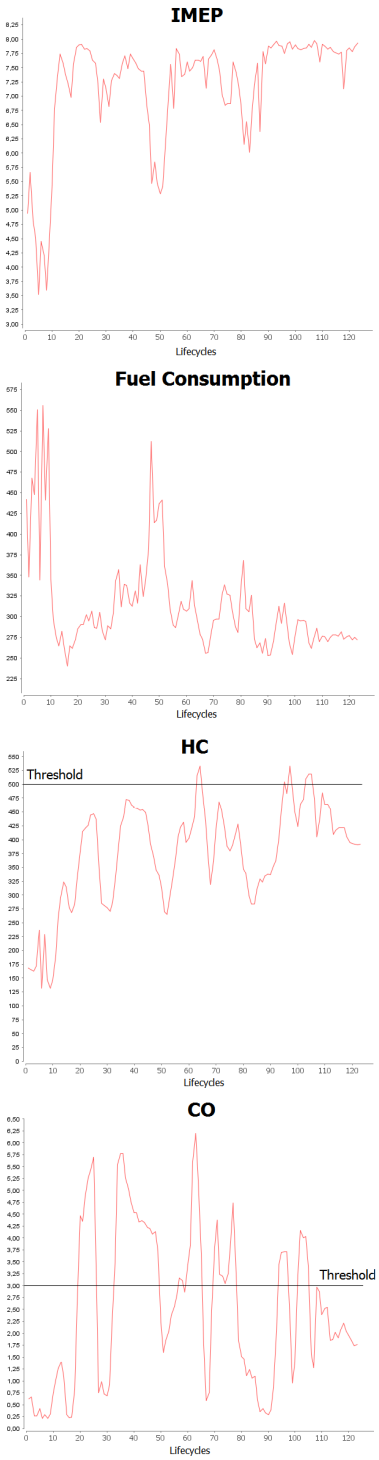

Figure 7: Torque and Fuel Consumption Optimization with Pollution Thresholds (outputs)

finds a value for each of the input parameters (IMF, ignition advance, and SOI) that maximizes the IMEP, minimizes the SFC, and keeps the HC and CO under their threshold.

During this test, ESCHER had to wait 20 seconds between each cycle for the gaz analyzer. Hence, the 123 cycles took 41 minutes. This is about twice faster than the classical optimization method.

7 CONCLUSION

In this paper, we presented an Adaptive Multi-Agent System, called ESCHER, which controls heat engines without a complete understanding of the controlled engine and basing its behavior on self-organizing agents. All the controller needs to know is how the engine behaves. In other words ESCHER is of black-box type: it only perceives the process inputs and outputs, but not its internal mechanisms. This property should make ESCHER generic enough to be easily applied to all kind of systems. Systems using similar types of agents have been applied to the control of the temperature of a bioprocess (Videau et al., 2011), and are currently being tested in the contexts of ambiant systems (Guivarch et al., 2012) and intelligent building energy management (Gatto et al., 2013).

The basic principle of our controller is the following: the multi-agent system memorizes the state of the process inputs/outputs when an action is applied and observes the reactions of the process. It will use this information 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 process, but it continually learns from its actions and manages to control the engine. Since the learning is parallel to the control, ESCHER continuously self-adapts to the process.

This learning ability is used in the context of automatic ECU calibration, as finding the right control actions on the engine is equivalent to finding the optimal ECU parameters. Our tool allows you to find the optimal settings in a relatively short period of time without putting a lot of means in practice and without detailed knowledge of the system.

Indeed, ESCHER does not need any prerequisite knowledge other than the intentions of the user (i.e. some criticality functions). It is also able to satisfy multi-criteria constraints on multiple inputs and outputs. Each time it performs an action, it learns from it, improving its control and adapting itself to the evolution of the process. Moreover, the independence between Controller Agents gives ESCHER a certain modularity. 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 without any knowledge about the rest of the controlling system.

The advantage of this approach (not having to perform a full scan of the parameters to find an optimum) can also be a downside, as the tool does not build a predictive model from an extensive base of acquired data that could be used later. Hence, our future work should focus on the extraction of information from the Context Agents for human users.

ACKNOWLEDGEMENTS

This work is part of the ORIANNE project, funded by the FUI (french acronym for single interministerial fund). It is lead by Aboard Engineering, and involves FH Electronics, IRIT, IRSEEM, CERTAM, CEVAA, and Renault.

Figure 8: ORIANNE Logo

REFERENCES

Astrom, K. J. and Hagglund, T. (1995). *PID Controllers: Theory, Design, and Tuning*. Instrument Society of America, Research Triangle Park, NC, second edition.

Colosimo, B. M. and Del Castillo, E., editors (2007). *Bayesian Process Monitoring, Control and Optimization*. Taylor and Francis, Hoboken, NJ.

Dabo, M., Langlois, N., Respondek, W., and Chafouk, H. (2008). 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.

Feldbaum, A. A. (1961). Dual control theory, I-IV. *Automation Remote Control*, 21-22.

Gatto, F., Gleizes, M.-P., and Elicegui, L. (2013). Saver: Self-adaptive Energy Saver. In *European Workshop on Multi-Agent Systems (EUMAS), Toulouse, France*.

Georgé, J.-P., Gleizes, M.-P., and Camps, V. (2011). Cooperation. In Di Marzo Serugendo, G., Gleizes, M.-P., and Karageogos, A., editors, *Self-organising Software*, Natural Computing Series, pages 7–32. Springer Berlin Heidelberg.

Guivarch, V., Camps, V., and Péninou, A. (2012). Context awareness and adaptation in ambient systems by an adaptive multi-agent approach. In *International Joint Conference on Ambient Intelligence, Italy*.

Hagan, M. T., Demuth, H. B., and De Jesus, O. (2002). An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control*, 12(11):959–985.

Jankovic, M. and Kolmanovsky, I. (2000). Constructive lyapunov control design for turbocharged diesel engines. *IEEE Transactions on Control Systems Technology*, 8(2):288 –299.

Lee, C. C. (1990). Fuzzy logic in control systems: Fuzzy logic controller. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418.

Lemouzy, S., Camps, V., and Glize, P. (2011). Principles and properties of a MAS learning algorithm: A comparison with standard learning algorithms applied to implicit feedback assessment. In *2011 International Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 228–235.

Malikopoulos, A. A., Assanis, D. N., and Papalambros, P. Y. (2009). Real-time self-learning optimization of diesel engine calibration. *Journal of Engineering for Gas Turbines & Power*, 131(2):22803.

Nikolaou, M. (2001). Model predictive controllers: A critical synthesis of theory and industrial needs. *Advances in Chemical Engineering*, 26:131–204.

Noël, V. (2012). *Component-based Software Architectures and Multi-Agent Systems: Mutual and Complementary Contributions for Supporting Software Development*. Phd thesis, Universit de Toulouse, Toulouse, France.

Stengel, R. F. (1991). Intelligent failure-tolerant control. *IEEE Control Systems*, 11(4):14–23.

Videau, S., Bernon, C., Glize, P., and Uribelarrea, J.-L. (2011). Controlling Bioprocesses using Cooperative Self-organizing Agents. In Demazeau, Y., editor, *PAAMS*, volume 88 of *Advances in Intelligent and Soft Computing*, pages 141–150. Springer-Verlag.