Policy search reinforcement learning for automatic wet clutch engagement

Matteo Gagliolo, Kevin Van Vaerenbergh, Abdel Rodríguez, Ann Nowé
CoMo, VUB (Vrije Universiteit Brussel), Pleinlaan 2, 1050 Brussels, Belgium
mgagliol@vub.ac.be
Stijn Goossens, Gregory Pinte, Wim Symens
FMTC (Flanders' Mechatronics Technology Centre), Celestijnenlaan 300D, 3001 Heverlee, Belgium
stijn.goossens@fmtc.be

***Abstract*— In most existing motion control algorithms, a reference trajectory is tracked, based on a continuous measurement of the system's response. In many industrial applications, however, it is either not possible or too expensive to install sensors which measure the system's output over the complete stroke: instead, the motion can only be detected at certain discrete positions. The control objective in these systems is often not to track a complete trajectory accurately, but rather to achieve a given state at the sensor locations (e.g. to pass by the sensor at a given time, or with a given speed). Model-based control strategies are not suited for the control of these systems, due to the lack of sensor data. We are currently investigating the potential of a non-model-based learning strategy, Reinforcement Learning (RL), in dealing with this kind of discrete sensor information. Here, we describe ongoing experiments with a wet clutch, which has to be engaged smoothly yet quickly, without any feedback on piston position.**

I. INTRODUCTION

RL problems [1] are a class of machine learning problems where an agent must learn to interact with an unknown environment, using a "trial and error" approach. At a given timestep t, the agent may execute one of a set of *actions* $a \in \mathcal{A}$, possibly causing the environment to change its *state* $s \in \mathcal{S}$, and generate a (scalar) *reward* $r \in \mathbb{R}$. Both state and action spaces can be multidimensional, continuous or discrete. An agent's behavior is represented by its *policy*, mapping states to actions. The aim of a RL algorithm is to optimize the policy, maximizing the reward accumulated by the agent. Simply stated, RL consists in learning from a teacher (the environment) who cannot tell us *what* to do next (the optimal policy), but only *how good* we are doing so far (the reward signal). It therefore offers a suitable tool for controlling systems with discrete sensor information. The target state at the discrete sensors location can be incorporated in the reward signal, favoring the desired behavior.

Two main families of RL approaches can be distinguished. In *value based* methods, the expected future reward $Q(s, a)$ allowed by taking action a in state s is estimated: the policy consists in selecting the action a which maximizes Q in the current state s. This can be done storing and updating estimates in tabular form, or resorting to function approximators if $\mathcal{A}$ and $\mathcal{S}$ are too big, or continuous. In *direct policy search*, the space of policies is searched for directly, maximizing the reward.

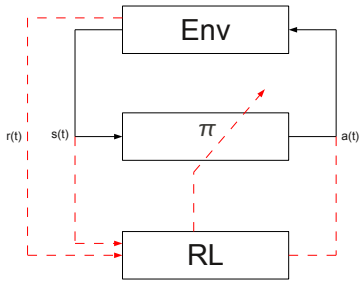

Fig. 1: Block representation of RL. Env: environment (plant being controlled). Π: policy (controller), generating the action (control signal) $a(t)$ given the current state of the plant $s(t)$; RL: learning algorithm, adapting the policy based on state, action, and reward signals. Continuous lines indicate online interactions, while discontinuous lines refer to the learning process, which can be online or offline, depending on the particular algorithm.

In both cases, learning is organized in a sequence of *epochs*, each consisting of a sequence of interactions with the environment. Another independent classification can be made among *offline* learning, where the policy is updated only after one or more epochs have been completed; and *online* learning, where such update can be performed also during an epoch (Fig. 1).

II. EXPERIMENTAL SETUP

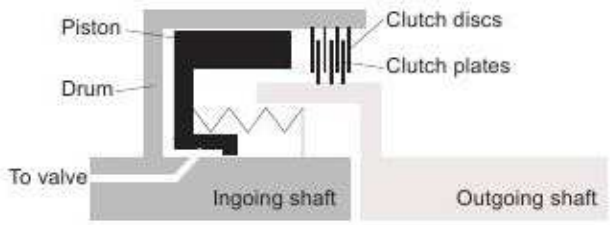

Fig. 2: Schematic representation of the wet clutch

The setup consists of a wet clutch (Figure 2), used in heavy duty transmission systems to engage/disengage differ-

ent loads. We considered in particular the automatic control of the engagement phase, which has to be fast enough, but also smooth, in order to avoid disturbing the operator. This latter requirement can be implemented minimizing the *jerk*, defined as the second derivative of the slip, which is the difference in angular speed among the two shafts.

The issue with this setup is that there is no sensor providing information about the actual position of the piston, which can only be detected when the piston actually touches the plates, and the slip phase begins. Given the available signals (oil pressure and temperature), there is no reference trajectory that can be tracked using classical control methods. In this sense, this setup is a good example of the discrete sensor systems mentioned above (see also [2]). Moreover, there currently is no reliable model of the whole engagement: an approximate model is available for the filling phase, until the piston touches the plates, but not for the following slip phase, which would allow to simulate and optimize the resulting jerk. However, it has been observed that the jerk depends on the speed of the piston when this reaches the plates: the lower the speed, the lower the jerk, and the smoother the engagement. In other words, there is a trade-off among the two objectives: on one hand, a very slow piston movement will result in a smooth engagement, which takes a long time; on the other hand, a fast piston will engage in a short time, but it will also cause a jerky engagement, and possibly damage the setup.

While the availability of a model is not in principle necessary for RL, it can in practice allow us to speed up the learning process, adopting an incremental approach. In a first phase, we can optimize the policy on the filling phase model, aiming at obtaining a fast engagement, but with a low piston velocity at engagement. In a second phase, we will transfer the obtained policy on the real clutch, and further improve its performance, reducing the actual jerk.

III. METHODS

Three *direct policy search* RL methods are being evaluated on this setup, all representing the policy as a probability density function (pdf) over actions, which is updated offline, after each epoch: the policy gradient (PG) method [3], where the actions are drawn from a Gaussian distribution; PG with parameter exploration (PGPE) [4], where the pdf is a mixture of Gaussians; and a continuous learning automaton (CARLA) [5], where the pdf over the actions is non-parametric. In the following three subsections, we briefly describe each method.

A. Policy gradients (PG)

In Policy Gradients (PG) methods [3], the policy is represented as a parametric probability distribution over the action space, conditioned by the current state of the environment. Epochs are subdivided into discrete time steps: at every step, an action is randomly drawn from the distribution, conditioned by the current state, and executed on the environment, which updates its state accordingly. After an epoch has been completed, the parameters of the policy are updated, following a Monte Carlo estimate of the expected cumulative (discounted) reward. In this section we describe the basic ideas behind this estimation, following [6], to which the reader can refer for further details.

To avoid cluttering notation, we consider a RL problem with a scalar action $a \in \mathcal{A} \subset \mathbb{R}$, and scalar state $s \in \mathcal{S} \subset \mathbb{R}$. Be $\boldsymbol{\theta} \in \mathbb{R}^d$ the d-dimensional parameter defining the conditional distribution over actions $\pi_{\boldsymbol{\theta}}(a \mid s)$, from which actions are drawn, and $r_t \in \mathbb{R}$ the reward obtained at timestep $t \in \mathbb{N}$. The index we intend to maximize is the cumulative discounted reward R_0, with $R_t = \sum_{k=t}^{T} \gamma^{t-k} r_k$ being the *return* at time t, where $\gamma \in (0, 1)$ is a discount factor, expressing a preference for early over late rewards, and T is the duration of an epoch. Be $H = \{s_0, a_0, \ldots, s_T, a_T\}$ the full history of state-action pairs, $H \in \mathcal{H}$, and $R(H)$ the corresponding return. The probability of observing a given history H depends on $\boldsymbol{\theta}$, on the unknown state transition probability $p(s_{t+1} \mid s_t, a_t)$, and on the unknown initial state probability $p(s_0)$, as

$$p(H) = p(s_0) \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t) \pi_{\boldsymbol{\theta}}(a_t \mid s_t). \qquad (1)$$

The expected value of R is defined as

$$J = E\{R(H)\} = \int_{\mathcal{H}} p(H) R(H) dH. \qquad (2)$$

The basic idea behind policy gradients is to follow the gradient of J with respect to $\boldsymbol{\theta}$, in order to maximize the expected return. While such gradient cannot be evaluated in closed form, it can be estimated as follows. Bringing the derivative operator inside the integral, we can write:

$$\nabla_{\boldsymbol{\theta}} J = \int_{\mathcal{H}} \left[\nabla_{\boldsymbol{\theta}} p(H) R(H) + p(H) \nabla_{\boldsymbol{\theta}} R(H) \right] dH. \qquad (3)$$

As the observed rewards do not depend on $\boldsymbol{\theta}$, but only on H, we have $\nabla_{\boldsymbol{\theta}} R(H) = 0$, therefore the second term of the addition can be discarded. The first term can be rewritten using the "likelihood ratio" trick, multiplying and dividing it by $p(H)$, thus obtaining

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J &= \int_{\mathcal{H}} p(H) \nabla_{\boldsymbol{\theta}} \log p(H) R(H) dH = \\ &= E\{\nabla_{\boldsymbol{\theta}} \log p(H) R(H)\}. \end{aligned} \qquad (4)$$

Given that (1) is a product, its logarithm can be expressed as a sum

$$\begin{aligned} \log p(H) &= \log p(s_0) + \sum_{t=0}^{T} \log p(s_{t+1} \mid s_t, a_t) + \\ &+ \sum_{t=0}^{T} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t), \end{aligned} \qquad (5)$$

with the advantage that deriving w.r.t. $\boldsymbol{\theta}$ leaves only the rightmost terms, which can be evaluated analytically as π is derivable. We can therefore perform a Monte Carlo estimate

of (4), after running N epochs, as follows

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J &= E\{\nabla_{\boldsymbol{\theta}} \log p(H) R(H)\} \approx \\ &\approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^n \mid s_t^n) R(H^n), \quad (6)\end{aligned}$$

where n is used as an index, to refer to the n-th epoch.

In practice, most authors suggest to use $N = 1$, updating $\boldsymbol{\theta}$ after each epoch [3], [6]. Moreover, to reduce the variance of the estimate, the term $R(H)$ is replaced by T returns R_t, each multiplied with the corresponding term $\log \pi_{\boldsymbol{\theta}}(a_t \mid s_t)$, thus discarding past rewards; and shifted by a baseline b, which, in the simplest case, is just the average return observed so far:

$$\nabla_{\boldsymbol{\theta}} J \approx \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t)(R_t - b). \quad (7)$$

Intuitively, following the gradient (7), the probability of taking action a_t in state s_t is increased if the return R_t, accumulated starting at time t, is larger than the baseline b, representing the average return, while it is decreased otherwise. Fig. 3 illustrates the effect of PG in a simple case, with no state information, $T = 1$, and Gaussian π, with $\boldsymbol{\theta} = (\mu, \sigma)$.

B. PG with parameter exploration (PGPE)

A major disadvantage of PG methods is that drawing a random action at every timestep may result in noisy control signals, as well as noisy gradient estimates. Moreover, the policy is required to be differentiable w.r.t. its parameters. To overcome these issues, PG with Parameter Exploration (PGPE) was introduced [7], [4]. In this method, the random sampling and policy evaluation steps are, in a sense, "inverted": the policy is a parametric function, not necessarily differentiable, therefore it can be an arbitrary parametric controller; the parameter value to be used is sampled at the beginning of each epoch from a Gaussian distribution, whose parameters are in turn updated at the end of the epoch, again following a Monte Carlo estimate of the gradient of the expected return. In other words, rather than searching the parametric policy space directly, PGPE performs a search in a "meta-parameter" space, whose points correspond to probability distributions over the (parametric) policy space.

Be again $\boldsymbol{\theta}$ the parameter of the policy f, where $a_t = f_{\boldsymbol{\theta}}(s_t)$; and $\boldsymbol{\alpha}$ the meta-parameter defining the distribution $p_{\boldsymbol{\alpha}}(\boldsymbol{\theta})$ over parameter values. With passages analogous to PG, we can again estimate the gradient of the expected return, this time as

$$\nabla_{\boldsymbol{\alpha}} J \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\alpha}} \log p_{\boldsymbol{\alpha}}(\boldsymbol{\theta}^n)(R_0^n - b), \quad (8)$$

where $\boldsymbol{\theta}^n$ is the parameter used at the n-th of the N epochs considered; also in this case, we can use $N = 1$. Compared with PG, in PGPE the information about the state-action pairs actually visited, conveyed by the history H, is lost, and $\boldsymbol{\theta}$ is evaluated directly based on the return R_0 obtained for the whole epoch. In the terminology of [8], PGPE should therefore be considered a *phylogenetic* method, as evolutionary computation, while PG is *ontogenetic*.

A multimodal PGPE was proposed in [9]: in this case, the distribution is a mixture of Gaussians,

$$p_{\boldsymbol{\alpha}}(\boldsymbol{\theta}^i) = \sum_{k=1}^{K} w_k^i \mathcal{N}(\mu_k^i, \sigma_k^i), \quad (9)$$

where K is the number of modes, $i = 1, \ldots, d$ denotes one of the d dimensions of the parameter, and the weights w_k^i are updated together with the parameters of each Gaussian, following an approximated gradient, with the additional constraint that $\sum k = 1^K w_k^i = 1$.

C. Continuous action reinforcement learning automata (CARLA)

Learning Automata [10] are simple reinforcement learners for single-stage, stateless environments, and discrete actions. In this kind of RL problems, which corresponds to the well-known multi-armed bandit problem [1], an epoch consists in the execution of a single action, followed by the observation of the corresponding reward. Also in this case, the policy is represented as a probability distribution over actions, which is updated after each epoch. An extensions to continuous actions, similar to PG, has been proposed in [11]; here, we adopt instead the alternative formulation of [12], termed Continuous Actions Reinforcement Learning Automata (CARLA). In this method, the distribution $f(a)$ over action space is non-parametric: its density, initially uniform, is stored explicitly for all points of an equally spaced grid. After each epoch, the probability of the action executed is reinforced, mixing the current density with a Gaussian function with constant standard deviation λ (termed *spreading rate*), centered in the action itself, whose amplitude is proportional to the observed reward:

$$f_{t+1}(a) = \gamma_t [f_t(a) + \beta_t(a_t) \alpha e^{-\frac{1}{2}(\frac{a - a_t}{\lambda})^2}] \; \forall a \in \mathcal{A}. \quad (10)$$

Figure 4 illustrates the learning mechanism of CARLA.

IV. APPLICATION TO SETUP

In this section we describe the design decisions taken to apply RL to the wet clutch setup.

State and Action Spaces The clutch is controlled by a single continuous signal, the current to the valve controlling oil pressure. In the model, the available signals are oil pressure and estimated piston position. Of these, only oil pressure is available on the real clutch, therefore we cannot make use of the estimated piston position as state variable. Initially, we will not consider any state information, and control the clutch in open loop. The utility of adding oil pressure is questionable, as this signal tends to follow the input signal. When moving to the real setup, we will include the slip signal.

Policy For the policy, we are considering an open loop controller, defined by four parameters, the same one in the evolutionary computation approach of [13] (Figure 5); a simplified two-parameter version of the same signal, obtained

by fixing two of the four parameters; and a closed loop controller, consisting of a recurrent neural network, along the lines of [6]. In both cases, the RL problem can be represented as a stateless, single stage problem, where an epoch lasts for a single timestep, at which the policy, represented by a multidimensional parameter which corresponds to the action of the reinforcement learner, is applied to the system for a given time. For PG, a multi-stage formulation is possible.

Reward Regarding the reward signal, this has to be a scalar, favoring both objectives at once. For the simulated model, we adopted a reward of the form $r = c/(v + c)$, where v is piston velocity at engagement, $c = 10^{-3}$ is a constant. Attributing this reward at engagement time t results in a return $R = \gamma^t c/(v + c)$, where $\gamma \in [0, 1)$ is a constant discount factor. Note that the term γ^t, typically used in RL to discount rewards, favors early engagements; while the term $c/(v+c)$ tends to 0 for large velocities, is 0.5 for velocity $v = c$, and it tends to 1 when v tends to 0. In practice, γ can be used to control the trade-off among the two objectives: a low value favors early engagements, while a high value increases the impact of the second term, favoring smooth engagements (low piston velocity). Figure 6 plots the return, along with the evolution of related signals in the simulated model. When moving to the real clutch, we will instead minimize the jerk g, again using a reward of the form $const/(g + const)$.

V. EXPERIMENTS

In this section we report example plots from experiments with Multimodal PGPE, with 2 and 3 modes, using the two parameter version of the open loop signal in Fig. 5. A learning rate of 0.4 was used. After preliminary tests with higher values, the discount rate γ was set to an intermediate value, 0.5, in order to obtain a good compromise among the two objectives, favoring a smooth yet prompt engagement. Each experiment was repeated 25 times, using different random seeds, each run lasting for 2000 epochs. All runs converged to a different set of parameters (a, d), but with a similar performance in terms of piston velocity and engagement time. Plotting the final parameters for each of the 25 runs (Figure 7) allows to identify a region of parameter settings which produces a similarly good behavior, corresponding to a "ridge" in the reward function.

To highlight the convergence of the algorithm, in Figures 8, 9 we report instead the evolution of the two objectives, with 2 and 3 modes respectively. Aggregated statistics for all 25 runs are reported in the form of box-plots, each box corresponding to a group of 100 consecutive epochs. From top to bottom, the graphs represent, respectively, the two objectives (piston velocity and engagement time), to be minimized, and the corresponding rewards, to be maximized.

VI. CONCLUSIONS

The first results obtained are promising, as PGPE manages to obtain a smooth engagement in a few hundred epochs. Ongoing research is aimed at analyzing the effect of parameter γ on the trade-off among the two objectives. We will then proceed incrementally, first using the parameters found in this phase to initialize (a, d) in the four parameter model, and adapting all four parameters further in simulation; then transferring the best obtained controllers to the real clutch, optimizing the actual objective, i.e., reducing jerk.

VII. ACKNOWLEDGMENTS

This work has been carried out within the framework of the LeCoPro project (grant nr. 80032) of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 1998.

[2] S. Goossens, G. Pinte, W. Symens, M. Gagliolo, A. Rodriguez, and A. Nowé, "Reinforcement learning for repetitive systems with discrete sensors," in *30th Benelux Meeting on Systems and Control — Book of Abstracts*. Gent, Belgium: Universiteit Gent, 2011, p. 149.

[3] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE Intl. Conf. on Intelligent Robotics Systems (IROS)*, 2006.

[4] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010.

[5] A. Rodríguez, R. Grau, and A. Nowé, "Continuous actions reinforcement learning automata. performance and convergence," in *Intl. Conf. on Agents and Artificial Intelligence (ICAART)*, 2011.

[6] D. Wierstra, A. Frster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of IGPL*, vol. 18, no. 5, pp. 620–634, 2010.

[7] F. Sehnke, C. Osendorfer, T. Rückstiess, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with Parameter-Based exploration for control," in *ICANN '08: Proceedings of the 18th international conference on Artificial Neural Networks, Part I*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 387–396.

[8] J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber, "Ontogenetic and phylogenetic reinforcement learning," *Künstliche Intelligenz*, vol. 23, no. 3, pp. 30–33, 2009.

[9] F. Sehnke, A. Graves, C. Osendorfer, and J. Schmidhuber, "Multimodal Parameter-exploring Policy Gradients," in *Intl. Conf. on Machine Learning and Applications*. IEEE, 2010, pp. 113–118.

[10] K. S. Narendra and M. A. L. Thathachar, "Learning automata - a survey," *IEEE Transactions On Systems, Man And Cybernetics*, vol. SMC-4, no. 4, pp. 323–334, 1974.

[11] M. A. L. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004.

[12] M. N. Howell, G. P. Frost, T. J. Gordon, and Q. H. Wu, "Continuous action reinforcement learning applied to vehicle suspension control," *Mechatronics*, vol. 7, no. 3, pp. 263 – 276, 1997.

[13] Y. Zhong, B. Wyns, R. D. Keyser, G. Pinte, and J. Stoev, "Implementation of genetic based learning classifier system on wet clutch system," in *30th Benelux Meeting on Systems and Control — Book of Abstracts*. Gent, Belgium: Universiteit Gent, 2011, p. 124.

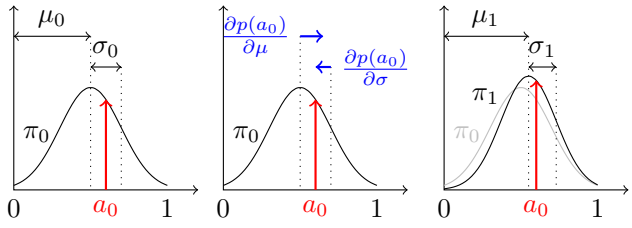

Fig. 3: A simple example illustrating the effect of one step of PG, with no state information, single stage epochs ($T = 1$), $\mathcal{A} = [0, 1]$, and Gaussian π, with $\boldsymbol{\theta} = (\mu, \sigma)$. *Left:* the first epoch is executed, drawing action $a_0 \sim \pi_0(a)$, and observing reward r_0, which in this case coincides with the return R_0. *Center:* as $R_0 > b$, following the gradient (7) increases $\pi(a_0)$. *Right:* updated distribution π_1, ready for the next epoch.

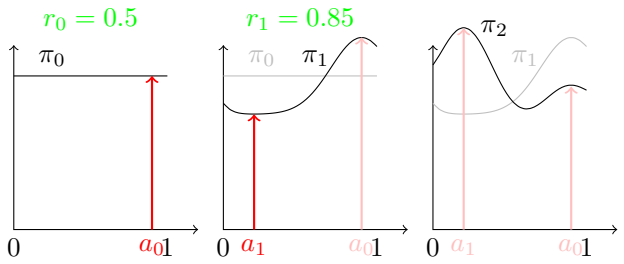

Fig. 4: An example illustrating CARLA. *Left:* the first epoch is executed, drawing action $a_0 \sim \pi_0(a)$, where π is initially uniform, and observing reward r_0. *Center:* π is updated, mixing it with a Gaussian bell centered in a_0, with amplitude proportional to r_0 as $R_0 > b$. An action a_1 is drawn from the updated π, resulting in a larger reward r_1. *Right:* π is updated, this time mixing with a larger bell, as the observed reward r_1 was bigger.

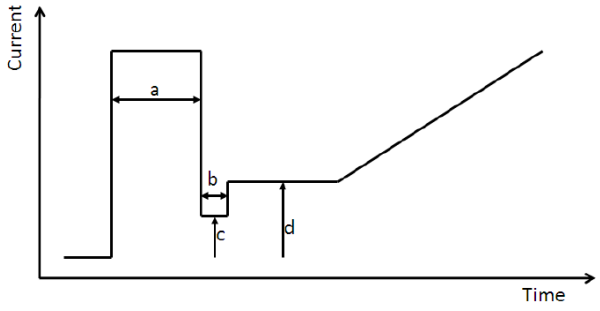

Fig. 5: Parametric input signal from [13], with four parameters (a, b, c, d). In our implementation, all parameter ranges are mapped to the unit interval $[0, 1]$. In the two parameter version, only a and d are free, while $b = 0$, and $c = d$.

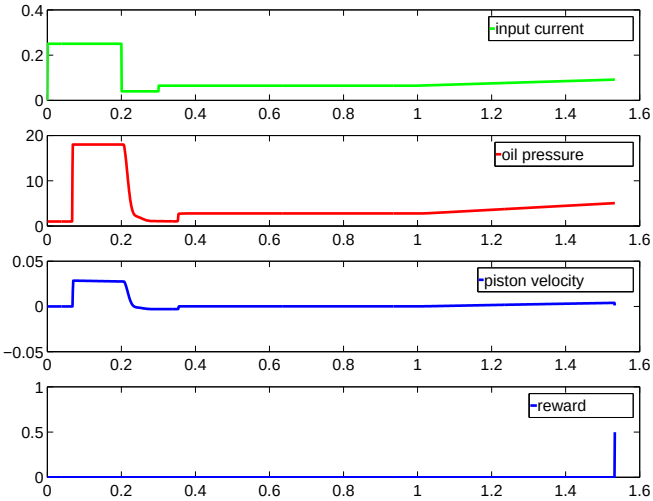

Fig. 6: A capture of the different signals available in the filling phase model of the wet clutch. Horizontal axes report time in seconds. From top to bottom: an example input signal (with $a = b = c = d = 0.5$); oil pressure; estimated piston position; reward signal, consisting of a single peak at engagement time, just above 1 second.

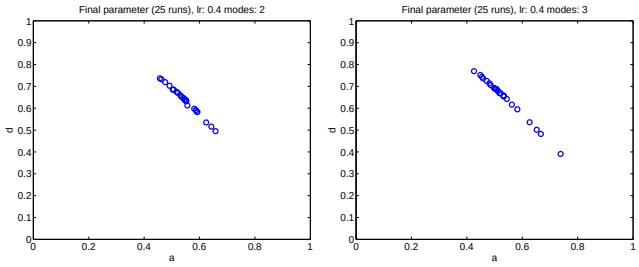

Fig. 7: PGPE with 2 (left) and 3 modes (right): Final parameter values for 25 runs, identifying a region of high reward.

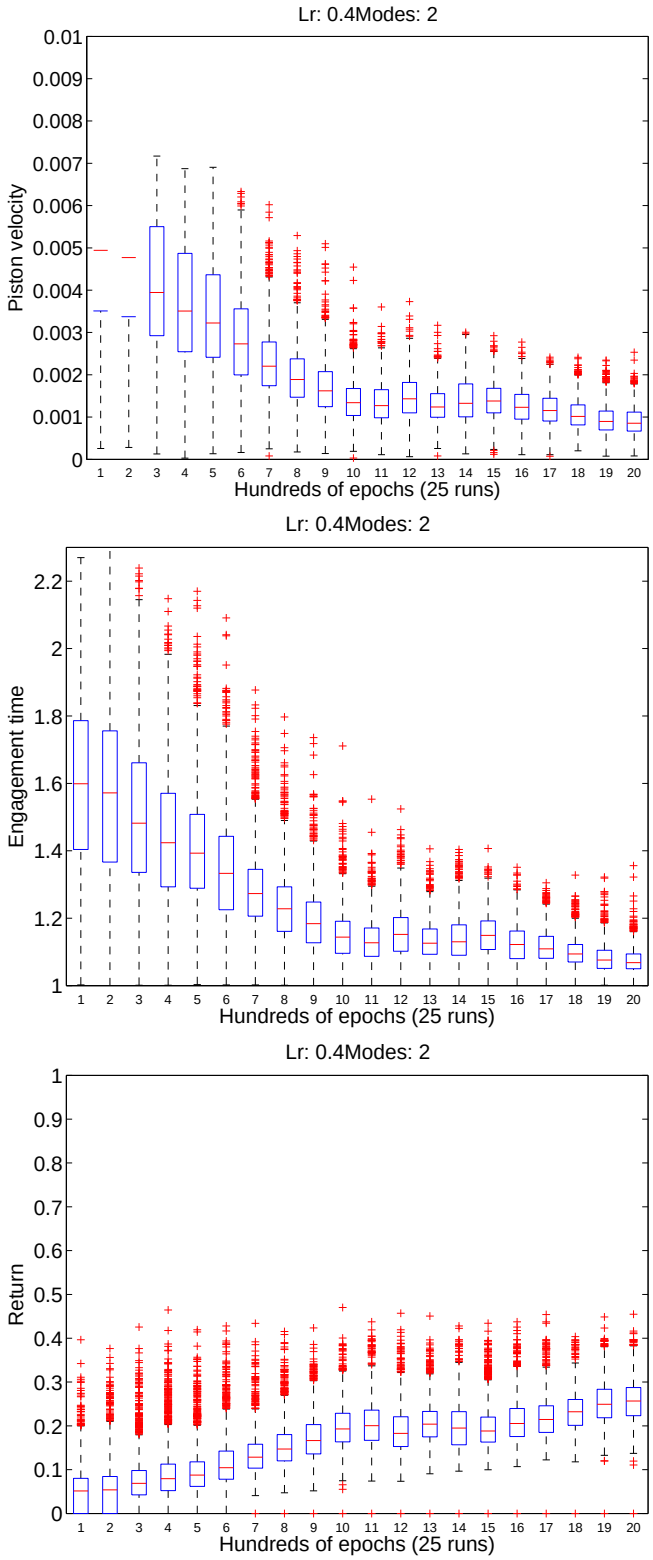

Fig. 8: PGPE with 2 modes: Performance for 25 runs. Each box corresponds to 100 consecutive epochs. Top: piston velocity, to be minimized. Center: engagement time, to be minimized. Bottom: reward, to be maximized.

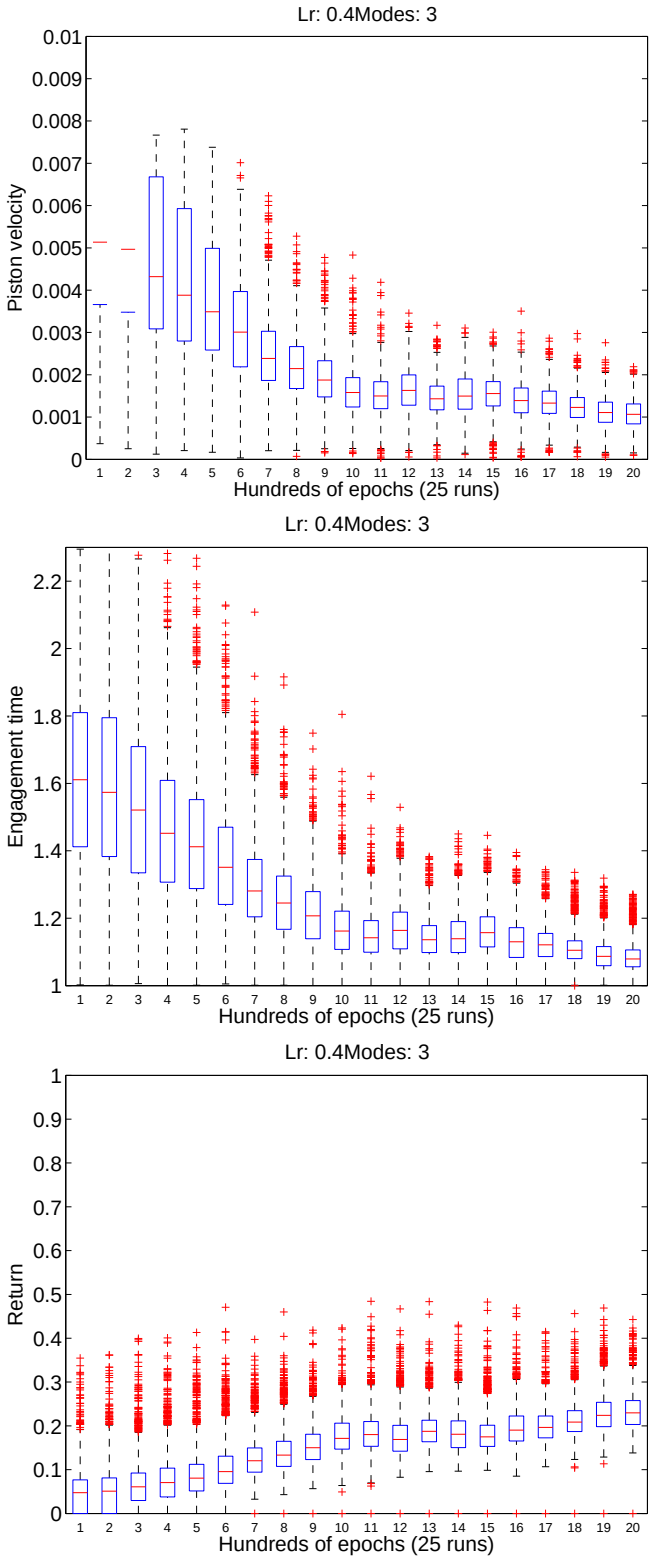

Fig. 9: PGPE with 3 modes: Performance for 25 runs, divided in bins of 100 epochs each. Top: piston velocity, to be minimized. Center: engagement time, to be minimized. Bottom: reward, to be maximized.