Abstract—A common approach when applying reinforcement learning to address control problems is that of first learning a policy based on an approximated model of the plant, whose behavior can be quickly and safely explored in simulation; and then implementing the obtained policy to control the actual plant. Here we follow this approach to learn to engage a transmission clutch, with the aim of obtaining a rapid and smooth engagement, with a small torque loss. Using an approximated model of a wet clutch, which simulates a portion of the whole engagement, we first learn an open loop control signal, which is then transferred on the actual wet clutch, and improved by further learning with a different reward function, based on the actual torque loss observed.

I. INTRODUCTION

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 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. Here we investigate the potential of a non-model-based learning strategy, Reinforcement Learning (RL), in dealing with this kind of limited feedback information. In control-theoretic terms, this paradigm consists in learning a control function iteratively, by interacting with the plant, without the need of a reference state signal to follow: the only feedback required is a scalar reward signal, whose magnitude is related to the quality of the controller. When a reliable model of the plant is available, the interaction can be repeated a large number of times in simulation, avoiding time-consuming, and potentially dangerous, experiments with the real plant.

In this paper, we describe experiments with a wet clutch, which has to be engaged smoothly yet quickly, without any feedback on the position of its moving parts. We extend work proposed in [1], which was solely based on a simulated model, presenting additional experiments, both in simulation and with a real clutch. Two approaches are compared: in a first set of experiments, an engagement policy is learned “from scratch” on the actual machine. In a second set, we first learn a policy based on the model, which simulates the initial phase of the engagement; the resulting policy is then transferred on the real machine, and improved with further learning. Such comparison is aimed at showing that, while a model of the plant is not a strict requirement for using RL, it can in practice speed up the learning process considerably, by reducing the number of experiments on the actual plant, and make it safer, by starting from a policy which is already adapted to the model.

The rest of the paper is organized as follows. The experimental setup is described in the next section. The following Section III briefly introduces RL methods, focusing in particular on PGPE [2], the RL algorithm used in the experiments. Section IV gives some general guidelines for applying RL to control problems, detailing the decisions taken in the two cases considered in this paper (simulated and real wet-clutch). Results are reported in Section V. Section VI concludes the paper, outlining directions for further research.

II. EXPERIMENTAL SETUP

The setup used in our experiments consists of a wet clutch, i.e. a clutch where the friction plates are immersed in oil, in order to smooth the transmission and increase the lifetime of the plates. Wet clutches are typically used in heavy duty transmission systems, such as those found in off-road vehicles and tractors. Fig. 1 offers a schematic representation of a wet clutch (from [3]).

![Fig. 1: Schematic representation of a wet clutch (from [3]).](image)
to the output shaft, which is connected to the wheels of the vehicle. The two sets of plates are free to translate axially, and a return spring keeps the clutch open: to close the clutch, the plates are pressed together by a hydraulic piston, which can be pushed against the plates by increasing the pressure of the oil in the clutch chamber. This pressure can in turn be controlled by the current input of an electromechanical servo-valve: by opening the valve, the clutch and supply line are filled up with oil, and the pressure increases until it is high enough to overcome the return spring force. As a result, the piston starts to move towards the plates. During this first part of the engagement, called the filling phase, no torque is transferred. The transmission only commences as the piston makes contact with the plates. The clutch then enters the slip phase, as the slip, defined as the difference in rotational speeds between the shafts, decreases. When the pressure is high enough, the output shaft is accelerated until it rotates synchronously with the input, and the slip reaches zero.

We consider in particular the automatic control of the engagement, which has to be fast enough, but also smooth, in order to avoid disturbing the operator, and reduce the wearing of friction plates. In principle, this smoothness requirement can be implemented by minimizing the jerk, defined as the second derivative of the slip: however the measurement of the slip is already quite noisy, and deriving it twice would result in an even noisier signal.

An alternative approach, which will be used in the following, is to minimize the maximum torque dip (or torque loss). The torque of the outgoing shaft, initially null when the clutch is open, increases to a maximum during engagement, due to a peak of angular acceleration; it then decreases after the engagement, converging to a stable value which depends on the angular velocity reached, and on a friction coefficient. It has been observed in practice that smoother engagements are also characterized by a smaller torque dip, therefore this quantity can be used as the control objective, to be minimized.

The main 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 limited feedback systems mentioned above (see also [1]). For this reason, wet clutches are commonly controlled in open loop, applying a signal which is either defined ad-hoc, or learned iteratively [4].

A parametric form of such a signal has been adopted in [5], for a genetic-based optimization approach, and it is displayed in Fig. 2. First, a large current pulse is sent to the valve, in order to generate a high pressure level in the oil chamber of the clutch, which will allow the piston to overcome the resistance of the preloaded return spring, and accelerate towards the friction plates. After this pulse, a lower constant current is sent out, in order to decelerate the piston as it approaches the friction plates: before reaching this low value, the current decreases shortly to an even lower value, creating a small dent in the signal, which should act as a “brake”, limiting the speed of the piston. Finally, the current signal grows following a ramp with a low slope, closing the clutch smoothly. The signal is parametrized as follows: the first parameter \((a)\) controls the duration of the initial peak, whose amplitude is fixed at the maximum level, while the last \((d)\) controls the low current level, just before the engagement begins. The remaining parameters \((b, c)\) control the shape of the “braking” dent, while the final slope during the engagement phase is fixed.

![Fig. 2: Parametric input signal from [5], with four parameters \((a, b, c, d)\). In our implementation, all parameter ranges are mapped to the unit interval \([0, 1]\).](image)

There currently is no reliable model of the whole engagement: an approximate model\(^1\) 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 torque loss. However, it has been observed that the torque loss depends on the speed of the piston when this reaches the plates: the lower the speed, the lower the torque loss, 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, with a large torque dip, 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 in simulation, on the filling phase model, aiming at obtaining a fast engagement, but with a low piston velocity: this objective can be assessed as the piston position is estimated as part of the modeling process. In a second phase, we will transfer the obtained policy on the real clutch, and further improve its performance, reducing the actual torque loss: in this phase, on the physical setup, the piston position is unknown, while the actual output torque can be measured.

\(^1\)Developed in Simulink\textsuperscript{\textregistered} by Julian Stoev, Gregory Pinte (FMTC), Bert Stallaert and Bruno Depraetere (PMA, Katholieke Universiteit Leuven).
### III. Methods

RL problems [6] 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 estimate the optimal policy, but only how good (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 limited feedback information: the target state at the sensor location(s) can be incorporated in the reward signal, favoring the desired behavior.

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; for example, in Policy Gradients (PG) methods [7], the policy is represented as a parametric probability distribution over the action space, whose parameters are updated following a Monte Carlo estimate of the expected cumulative (discounted) reward.

Two main families of RL approaches can be distinguished. In value based methods, the expected future reward \( Q(s, a) \) 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; for example, in Policy Gradients (PG) methods [7], the policy is represented as a parametric probability distribution over the action space, whose parameters are updated following a Monte Carlo estimate of the expected reward.

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. 3).

In this paper, we apply an existing variant of the basic PG, called policy gradients with parameter exploration (PGPE) [2]. In this approach, the parameters of a controller are adapted based on the return collected during the whole epoch, regardless of the trajectory in the state space. The advantage of using a direct policy search method is that it allows to implement in a straightforward manner the incremental approach described above: a policy that has been optimized based on the performance on the simulated plant can be used as a good starting point for learning to control the real plant.

In the remainder of this section we briefly describe PGPE, referring the reader to [2], [1] for further details.

In Policy Gradients (PG) methods, 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.

![Fig. 3: Block representation of RL. Env: environment (plant being controlled). II: 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.](image)

![Fig. 4: A simple example illustrating the effect of one step of PGPE, with no state information and single stage epochs (\( T = 1 \)). A single policy parameter \( \mathcal{A} = [0, 1] \) is sampled from a Gaussian prior \( \pi \), with \( \theta = (\mu, \sigma) \). Left: the first epoch is executed, drawing a parameter value \( a_0 \sim \pi_0(a) \), and observing a return \( R_0 \). Center: as \( R_0 > b \), following the gradient (1) increases \( \pi(a_0) \). Right: updated prior \( \pi_1 \), ready for the next epoch.](image)
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.

To simplify notation, we consider a parametric policy \( f_\theta \) with a scalar parameter \( \theta \). Be \( \alpha = (\mu, \sigma) \) the meta-parameter defining the Gaussian distribution \( p_{\alpha}(a) \) over parameter values. The index we intend to maximize is the expected value of the return \( R \) given \( \alpha \), \( J = E[R|\alpha] \). The gradient of this expected return \( J \) with respect to the metaparameter \( \alpha \) can be estimated as follows (see [2] for details):

\[
\nabla_\alpha J \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\alpha \log p_{\alpha}(a^n)(R^n - b),
\]

where \( \theta^n \) is the parameter used at the \( n \)-th of the \( N \) epochs considered (typically \( N = 1 \)), and \( b \) is a baseline return, which, in the simplest case, is the average return observed so far. Fig. 4 illustrates the effect of a single step of PGPE.

Compared with PG, in PGPE the information about the state-action pairs actually visited is lost, and \( \theta \) is evaluated directly based on the return \( R \) obtained for the whole epoch. In the terminology of [9], PGPE should therefore be considered a phylogenetic method, as evolutionary computation, while PG is ontogenetic.

IV. Application To Setup

In this section we describe the application of PGPE to control the wet clutch, both simulated and real. In general, in order to apply an RL method to solve an existing control problem, a number of design decisions have to be taken, each liable to have an impact on the performance of the algorithm. First, the state and action spaces of the problem need to be clearly defined: their dimensionality should be kept to a minimum, including only the most relevant variables, in order to allow the RL agent to discriminate among different situations, and different behaviors, while keeping the size of both spaces reasonable.

A related decision is the choice of the policy, which depends on the kind of RL algorithm used. For policy gradient methods, the policy can be an arbitrary parametric function of the state. As a simpler alternative, one can discard the state information entirely, and adopt instead an open loop approach, defining a parametric signal to be applied to the plant. In this case, the RL problem is reduced to a simpler optimization problem, in which the parameter of the signal needs to be optimized.

Last, a reward function has to be chosen. The purpose of this step is to translate the complex behavior of the learning agent into a scalar signal which can be reliably used to evaluate its performance. In general, the reward can be a function of time \( r(t) \): what is then optimized is the return, defined as the discounted cumulative reward \( R = \sum_{t=0}^{T} \gamma^t r(t) \), with \( \gamma \in (0, 1] \); in the simplified setting considered in this paper, a scalar return \( R \) can be attributed directly for the whole epoch. Designing a good reward function is not trivial, and it depends on the application. If a single, measurable objective \( y \) has to be optimized, the reward \( r(y) \) can simply be a monotonic function of the objective: it may still be useful to use a nonlinear function to keep \( r(y) \) to reasonable values. If multiple, contrasting objectives are present, the reward function should explicitly balance the objectives, translating them into a single scalar. If constraints on some objectives have to be enforced, this can be simply implemented by “punishing” constraint violations with a negative reward.

In the following, we give concrete examples of these steps in the two scenarios considered (simulated and real clutch engagement). While the parametric forms used may be applied in a different context, the numerical values proposed are specific to the actual setup which was used for the experiments, described in the following section.

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. The utility of adding oil pressure is questionable, as this signal tends to be heavily dependent on the input signal. We will therefore not consider any state information, and control the clutch in open loop in both scenarios.

Policy For the policy, we consider the open loop controller described above (Section II), defined by four parameters. 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.

![Fig. 5: 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 \( r(\theta) \), consisting of a single peak at engagement time, just above 1 second.](image-url)
**Reward** Regarding the reward signal, this has to be a scalar, favoring both objectives at once. For the simulated model, we adopted a reward based on piston velocity, of the form $r(v) = k/(v + k)$, where $v$ is piston velocity at engagement, $k = 10^{-3}$ is a constant. Attributing this reward at engagement time $t$ results in a return which is a function of both objectives, $R(v, t) = \gamma^t k/(v + k)$, where $\gamma \in [0, 1]$ is a constant discount factor. Note that the term $\gamma^t$, typically used in RL to discount rewards, favors early engagements; while the term $k/(v + k)$ tends to small values of $v$, is $0.5$ for velocity $v = k$, and it tends to $1$ when $v$ tends to $0$. This expression for $r$ is practical in this case, as $v$ can vary from about one order of magnitude among a good and a bad engagement: the saturating behavior of $r$ allows to discriminate easily among different engagements, while keeping the reward to a reasonable size. In practice, $\gamma$ 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). Fig. 5 plots the return, along with the evolution of related signals in the simulated model.

On the real setup, we can directly minimize the maximum torque loss $g$. Preliminary experiments using a reward of the form $k/(g + k)$ did not give satisfactory results, as in this case the observed values for $g$ remain in the same order of magnitude (typically between 200 and 1000). We then opted for a reward $r(g) = \exp k_1(1 - g/k_2)$, with $k_2 = 300$, corresponding to what can be considered a good value for the torque loss. Regardless of the value of $k_1$, this function will give a reward $r = 1$ for a torque loss of $g = k_2 = 300 Nm$, and $r > 1$ and $r < 1$, respectively, for smaller (i.e. better) and larger (i.e. worse) torque losses. The constant $k_1$ controls the steepness of the reward: after displaying the curve for different values of this parameter, we chose $k_1 = 3$, which seemed to give a reasonable steepness to the reward function. Note that both parameters were chosen heuristically: the important thing is just that the reward function is monotonically decreasing in the objective which we intend to minimize, the torque loss $g$, and it has reasonable values, and a reasonable steepness, on the expected range of such objective. To take into account engagement time, also in this case we discount the reward with a discount factor $\gamma$, obtaining a return $R(g, t) = \gamma^t r(g)$.

Note that, in both cases, the resulting policy will reflect the chosen compromise among the two objectives. If this changes slightly, and the reward function is modified accordingly, it may be possible to tune the policy according to the new reward; if the change is a drastic one, it may be better to reset the policy, and start learning from scratch.

**V. Experiments**

**A. Filling phase model**

In this subsection we report the results obtained with a MATLAB® implementation of PGPE, using the open loop signal in Fig. 2 on a Simulink® model of the filling phase. A learning rate of 0.8 was used, while the discount rate $\gamma$ was set to 0.9, in order to favor smooth over prompt engagements. Each experiment was repeated 25 times, using different random seeds, each run lasting for 200 epochs. To highlight the convergence of the algorithm, in Fig. 6 we report the evolution of the two objectives. As the policy parameter that is tested at each epoch is obtained by adding noise to the metaparameters of PGPE, the resulting learning curves appear to be very noisy: we therefore display aggregated statistics for all 25 runs, reported in the form of box-plots, each box corresponding to a group of 10 consecutive epochs. From top to bottom, the graphs represent, respectively, the two objectives (piston velocity and engagement time), to be minimized, and the corresponding reward, to be maximized. It can be seen that both objectives tend to stabilize, after about 70 epochs, greatly improving over the initial engagements.

**B. Real clutch**

In this subsection we report results obtained on the experimental test bench shown in figure 7, available at FMTC. It consists of a transmission containing the wet clutch which is to be controlled. An induction motor is connected to drive the system, while the combination of a transmission and a flywheel are used to vary the load observed by the controlled transmission. A sensor is installed to measure the transferred torque. The experiments are performed with an engine that is controlled at a fixed speed, while at each epoch the output is accelerated from zero up to synchronous speed by engaging the clutch for first gear in the controlled transmission. All calculations are performed in MATLAB®, using MLIB to communicate with a dSPACE® 1103 control board.

At each epoch, the signal is generated according to the parameter values selected by PGPE. To avoid damaging the clutch, the resulting signal is first applied to close the clutch without a load: if the torque loss is already large, we can expect a much larger one with the load, therefore the signal can be discarded without further evaluation. More precisely, if the measured torque loss is above 900 Nm, then the epoch ends, objectives are set at very poor values ($g = 5000 Nm$, $t = 5 s$), and the return $R$ is set to 0. Otherwise, the signal is applied again with a load, this time returning the actual values for $g$, $t$, and the corresponding return $R$ for the epoch.

Fig. 8 reports results from a single run of PGPE, lasting 100 epochs. Each bar corresponds to an epoch: the first two plots report engagement time and torque loss observed, with artificially high values for unsuccessful engagements, while the lower plot reports the corresponding returns. Recall that the parameter tested in each epoch is a noisy version of the meta-parameter $\mu$ of PGPE, which changes slowly during the run. While the performance is generally poor, we can see a reduction of the frequency of engagements that had to be discarded (corresponding to bars exceeding the plot range), as well as a moderate improvement of both objectives, which start giving a relevant return around the 70th epoch: the fact that the performance remains noisy indicates that the $\sigma$ is still far from zero, therefore the actual parameter used is a quite noisy version of $\mu$, and the algorithm is still far from converging. To highlight the mild improvement along the run,
in Fig. 10, 11 we display two groups of three graph each, corresponding to two different engagements. The three plots in each group report, from top to bottom, the evolution of three quantities during a single engagement: the input current, the speed of the output shaft, and the relative torque, whose maximum value corresponds to the torque loss, the objective to be minimized along with engagement time. The top group represent the engagement at the first epoch: in this case the engagement is both slow and jerky, with a high torque loss. The lower group corresponds to the best engagement of the run, at epoch 75, with a torque loss of about 250: it can be seen from Fig. 8 that this particular engagement was an exception.

In the following experiment, we use the $\mu$ learned based on the filling phase model (see previous subsection) to initialize PGPE, as well as a smaller $\sigma = 0.2$ to avoid excessive exploration around this value. Fig. 9 reports the evolution of the objectives: this time the algorithm converges, in about 30 epochs, finding parameter values which allow for a very smooth engagement, with a small torque loss. Fig. 12 and 13 report two engagements from this run, at epochs 1 and

---

**Fig. 6:** PGPE, model-based experiments. Performance for 25 runs, divided in bins of 10 epochs each. Top: piston velocity, to be minimized. Center: engagement time, to be minimized. Bottom: reward, to be maximized.

**Fig. 7:** Setup used in the experiments (from [3]).

**Fig. 8:** PGPE on the real clutch, starting from scratch: evolution of engagement time (above), torque loss (center), and reward (below) during learning.

In the following experiment, we use the $\mu$ learned based on the filling phase model (see previous subsection) to initialize PGPE, as well as a smaller $\sigma = 0.2$ to avoid excessive exploration around this value. Fig. 9 reports the evolution of the objectives: this time the algorithm converges, in about 30 epochs, finding parameter values which allow for a very smooth engagement, with a small torque loss. Fig. 12 and 13 report two engagements from this run, at epochs 1 and
30, respectively. In this case the initial engagement is already relatively good. This, along with the fast convergence to a good signal, is also an indirect confirmation of the goodness of the filling phase model.

![Graph showing engagement time, torque loss, and reward evolution during learning.](image)

**Fig. 9:** PGPE on the real clutch, with initialization: evolution of engagement time (above), torque loss (center), and reward (below) during learning.

**VI. Conclusions**

We presented a practical application of RL, where an open-loop control signal for a wet clutch is learned, aimed at obtaining smooth engagements. Our experiments show that the availability of a software model, albeit a partial one, can allow to learn a good initial policy in simulation, greatly reducing the convergence time of subsequent experiment on the actual machine.

Ongoing research is aimed at analyzing the effect of parameter $\gamma$ on the trade-off among the two objectives, and at comparing PGPE with alternative iterative approaches. The longer term goal of our research is to develop a practical method for adapting the control signal to changing conditions. To this aim, we will consider more complex policies, introducing some basic state information (e.g. oil temperature).

**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**


Fig. 10: Experiments on the real clutch, without initialization: engagement at epoch 1. The three plots report, from top to bottom: current to the valve (input signal), speed of the output shaft, and the relative torque, whose peak value corresponds to the torque loss (objective to be minimized).

Fig. 12: Experiments on the real clutch, with initialization: engagement at epoch 1. The three plots report, from top to bottom: current to the valve (input signal), speed of the output shaft, and the relative torque, whose peak value corresponds to the torque loss (objective to be minimized).

Fig. 11: Experiments on the real clutch, without initialization: engagement at epoch 75. The three plots report, from top to bottom: current to the valve (input signal), speed of the output shaft, and the relative torque, whose peak value corresponds to the torque loss (objective to be minimized).

Fig. 13: Experiments on the real clutch, with initialization: engagement at epoch 30. The three plots report, from top to bottom: current to the valve (input signal), speed of the output shaft, and the relative torque, whose peak value corresponds to the torque loss (objective to be minimized).