Received July 27, 2020, accepted August 6, 2020, date of publication August 10, 2020, date of current version August 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3015576

Reinforcement Learning Based Control Design for a Floating Piston Pneumatic Gearbox Actuator

TAMÁS BÉCSI[1], (Member, IEEE), ÁDÁM SZABÓ[1], BÁLINT KŐVÁRI[1], SZILÁRD ARADI[1], (Member, IEEE), AND PÉTER GÁSPÁR[2]

[1]Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, 1111 Budapest, Hungary
[2]Systems and Control Laboratory, Institute for Computer Science and Control, 1111 Budapest, Hungary

Corresponding author: Tamás Bécsi (becsi.tamas@mail.bme.hu)

This work was supported in part by the Higher Education Excellence Program in the frame of Artificial Intelligence research area of Budapest University of Technology and Economics (BME FIKP-MI/FM), and in part by the Hungarian Government and co-financed by the European Social Fund (Talent management in autonomous vehicle control technologies) under Grant EFOP-3.6.3-VEKOP-16-2017-00001.

ABSTRACT Electro-pneumatic actuators play an essential role in various areas of the industry, including heavy-duty vehicles. This article deals with the control problem of an Automatic Manual Transmission, where the actuator of the system is a double-acting floating-piston cylinder, with dedicated inner-position. During the control design of electro-pneumatic cylinders, one must implement a set-valued control on a nonlinear system, when, as in the present case, non-proportional valves provide the airflow. As both the system model itself and the qualitative control goals can be formulated as a Partially Observable Markov Decision Process, Machine learning frameworks are a conspicuous choice for handling such control problems. To this end, six different solutions are compared in the article, of which a new agent named PG-MCTS, using a modified version of the ''Upper Confidence bound for Trees'' algorithm, is also presented. The performance and strategic choice comparison of the six methods are carried out in a simulation environment. Validation tests performed on an actual transmission system and implemented on an automotive control unit to prove the applicability of the concept. In this case, a Policy Gradient agent, selected by implementation and computation capacity restrictions. The results show that the presented methods are suitable for the control of floating-piston cylinders and can be extended to other fluid mechanical actuators, or even different set-valued nonlinear control problems.

INDEX TERMS Intelligent agents, machine learning, pneumatic actuators, reinforcement learning, supervised learning, system testing.

I. INTRODUCTION

The popularity of autonomous functions is continuously increasing in the heavy-duty vehicle industry. Along with advanced driver assistance systems (ADAS), they are expected to increase fuel efficiency and decrease emissions while also enhancing safety [1]. Typical examples of these systems are platooning [2], automated highway driving [3], and autonomous yard maneuvering [4], having large influence on the future of transportation systems [5]. Most of these functions require object, or lane detection algorithms, where beside classical algorithms, Machine Learning (ML) is also widely used, such as the methods presented in [6]–[8], and [9]. Machine learning also spreads rapidly in the field of vehicle control, such as lane-keeping [10] and steering control [11], [12]. On the other hand, machine learning algorithms also perform well in mechatronic systems, where the controlling aim is different and easier in some sense. Modeling the environment is more straightforward in these cases, though it remains nonlinear. In the case of electro-pneumatic actuators, the primary sources of nonlinearities come from the air's friction and compressibility. Despite these, they are widely used in robotics and heavy-duty vehicles, as they have high power density, simple maintainability, and high operational safety. There is an unlimited supply of air to be used and exhausted in the environment afterward, causing the fluid return lines to become unnecessary. While one of the most significant advantages of pneumatic systems is the

The associate editor coordinating the review of this manuscript and approving it for publication was Firooz B. Saghezchi.

compressibility of air, from control aspects, it is also one of its main drawbacks because it results in nonlinear behavior and adds delay to the system.

A transmission system's primary function is to extend the highly limited angular velocity and torque range of the internal combustion engine (ICE) to a much wider interval, which is needed during the operation of a vehicle. It forwards the power of the ICE to the wheels of the vehicle, hence it has a significant effect on the vehicle's emission and fuel efficiency, whose improvement is a vital goal of the vehicle industry. As a consequence, much research focuses on the optimization of shifting strategies. In [13] a Dynamic Programming based optimal gear shifting control strategy is proposed for a vehicle equipped with a Power-Shift Automated Manual Transmission (AMT), while [14] deals with the high-precision synchronization of clutchless AMT systems. Meanwhile, other researches focus on the control of the clutch and gearbox actuators.

In heavy-duty vehicles, mostly single- and double-acting pneumatic actuators are used as clutch and gearbox actuators or brake cylinders. While in robotics, many researchers focus on the modeling and control of Pneumatic Artificial Muscles (PAM), which show promising results in applications, where the interaction between humans and robots is crucial. Since they are used in more and more complex and often safety-critical applications, the importance of the accurate modeling and control of these systems is increasing.

Regarding the improvement of pneumatic models, a key challenge is the mathematical description of the hysteretic aspects of pneumatic actuators caused by friction. In [15] a Preisach model is proposed, which is only accurate in for a narrow displacement range. A generalized Bouc–Wen model [16] is also proposed, which shows a lower root mean square tracking error than the widely used Maxwell-slip model [17]. In [18] an empirical approach is proposed, where a fourth-degree polynomial is used in which the coefficients linear functions of the pressure, which proved to be more accurate, than most of the analytical methods.

There are many different position control methods proposed for pneumatic systems in the literature. In [19], a Linear Quadratic servo control is presented to control a single-piston pneumatic gearbox actuator, and PID type controllers are also commonly used due to their simplicity and low calculation cost. Still, they are often combined with other methods to enhance their performance. In [20], the controller is divided into a fuzzy-model-based controller, which cancels the effects of the nonlinearities. Hence the servo-based portion can be controlled by a linear PID controller. In [21], an adaptive fuzzy PD controller is combined with an integral branch, and a fuzzy inverse model is used to adjust the PD part dynamically. To reduce the modeling errors caused by the simplified modeling of the system dynamics and to increase the controller's performance, an active model-based, advanced nonlinear PID controller is proposed in [22].

Nonlinear and post-modern control techniques show promising results and possibly better performance, but they have higher calculation cost, which can be an issue in certain applications. In [23] a discrete-valued Model Predictive Controller (MPC) is developed to control a hybrid pneumatic-electric actuator. [24] presents and compares a discrete-valued MPC controller and a Sliding Mode Controller (SMC). Both MPCs [25] and SMCs [26] are usually combined with Pulse-Width Modulation (PWM) to achieve binary control of the solenoid valves. These solutions use a high number of solenoid valve activations, which has negative effects on the actuators' lifetime. In [27], a switched backstepping controller is presented, which shows better performance compared to PWM-based methods and uses less solenoid valve switches. In [28], the control of a hybrid actuator is presented, which consists of a pneumatic actuator controlled by 3/2 valves and a DC motor. High-quality position control of the pneumatic actuator is presented using discrete-values MPC method. At the same time, the authors also significantly reduced the valve switches compared to PWM-based methods. In [23], the proposed controller for the pneumatic cylinder is augmented with a payload estimation algorithm, and it presents experimental results. In [29], an SMC controller is developed for an electro-pneumatic clutch actuator, then controller reduction possibility is presented. A general solution is shown in [30], where an observer-based adaptive finite-time tracking control strategy is developed by combining dynamic surface control (DSC) technique and backstepping approach for a class of nonlinear systems. Control solutions for PAMs often use proportional valves, where it is possible to control for a constant mass flow rate. However, in the case of 2-way 2-position valves used in clutch actuators, the continuous control signal has to be discretized to provide binary control signals for the valves. In certain gearbox actuators, the problem becomes more complicated, using 3-way 2-position valves instead of 2-way 2-position valves since they can be used both for load and exhaust purposes, which is beneficial regarding the cost of these systems. Although they are not able to hold constant pressure within the chambers, thus the system becomes unstable.

As shown in literature, position control of a pneumatic actuator is achieved both by classical and post-modern control algorithms. However, each has different drawbacks and limitations which must be taken into account along with the system's properties and the control requirements, when the control method is selected, and the control structure is designed. One possible choice would be a linear controller, but as the system is highly nonlinear, either gain scheduling approach or linearization techniques must be applied. Still, they are expected to have limited performance compared to post-modern control techniques. Methods such as linear parameter varying (LPV) control, SMC, and MPC are able to overcome the fast and unstable system dynamics and the delayed effect of the control signals. As gearbox actuators are mostly operated by 3-way 2-position valves, the controller must provide binary control signals for the solenoid valves, hence these methods are often combined with PWM,

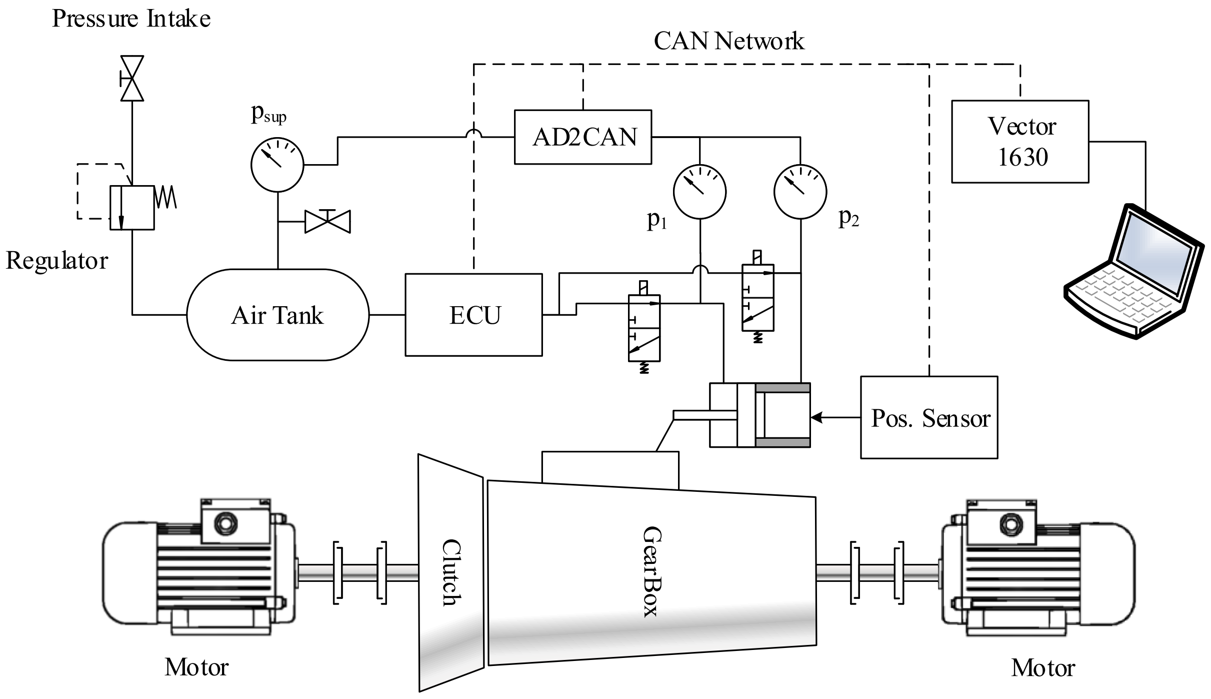

FIGURE 1. Schematic layout of the electro-pneumatic gearbox actuator.

although it cannot be used due to the strict limitation of the valve switches. Other discretization techniques and expensive control weighting can also be used, but they reduce the control accuracy.

While it is possible to find a trade-off between the quality of the gear change and the low number of valve switches, Artificial Intelligence (AI) based methods should also be considered, as they can handle complex environments and different goals, and often find strategies and correlations that are not possible with conventional methods.

Gear-change is an event-driven task, and the shift has a given time limit, and a well-defined goal, which makes this problem ideal for Reinforcement Learning (RL) based algorithms. Previous research showed promising results in the field [31], [32], leading to more extensive research presented in this article.

This paper aims to compare different ML algorithms through the control of an electro-pneumatic gearbox actuator. The designed algorithms include a Q-learning agent, a Policy Gradient (PG) agent, a Supervised Learning (SL) agent, and combined algorithms utilizing the synergy between the Monte-Carlo Tree Search (MCTS) and different reinforcement learning techniques by integrating planning into learning (Planning agent) and integrating planning into prediction (PG-MCTS) which formulation is the main contribution of this research paper. A standalone MCTS algorithm is also provided, even though it is not suitable for real-time use. Though, it serves as the training data source of the supervised learning agent and as a benchmark for the other algorithms. Concerning the results of the comparison and the limitations of a commercial actuator control unit, the neural network agent is trained by the Policy Gradient algorithm on the model, which is implemented and tested on the real system.

The paper is organized as follows: Section II introduces the nonlinear model of the actuator and the control goals. Section II-A presents the training environment designed to conform with the Markov Decision Process (MDP) paradigm, including the observable state space, the action space, and the reward function. Section III presents the designed algorithms, along with the parameters used for training. Section IV provides the comparison of the agents' performances and strategy choice using simulation results. Section V describes the real test rig, where an automotive electronic control unit (ECU) and a heavy-duty gearbox are used for testing, and also presents and evaluates the measurement results and Section VI provides some concluding remarks.

II. SYSTEM DESCRIPTION

Fig. 1 shows the schematic overview of the system. The system consists of a heavy-duty gearbox, operated by the shift actuator. A clutch is installed on the gearbox, and the two axles are connected to electric motors, which ensure the rotations for testing. The control of the actuator is performed by an automotive ECU, which collects the measurements and controls the two valves through binary commands. A laser position sensor measures the endpoint of the piston rod, while the pressure sensors are collecting data from the air tank, and the two working chambers of the piston. The data logging is performed via CAN network.

The examined shift actuator is realized with a double-acting floating-piston cylinder shown as the model part of Fig. 3. It needs to switch between three dedicated positions, namely High (H), Low (L), and Neutral (N), which are the two end positions of the cylinder, and an intermediate position, which limits the movement of the floating piston to one side of the cylinder, respectively. The system model consists of two solenoid valves, two pistons, three chambers, a detent mechanism, and a Shift Finger. The detent mechanism is used to fixate the main piston in the dedicated positions and to prevent its unintentional movement.

Position control is realized by 3-way 2-position valves, connecting Chamber 1 and Chamber 2 either to the pressure supply or to the environment for exhausting. The third chamber is the control chamber, which only has an exhaust port, and serves as an air spring. The floating piston tunes the volume and through it the pressure of the control chamber, which then reacts to the movement of the floating piston, also affecting the movement of the main piston indirectly. The main piston also has a cross-section area adjacent to the control chamber. Therefore it also has a direct effect on its movement.

The main piston actuates the gear change. As it moves due to the force generated by the chamber pressures, it rotates the shift finger, which shifts the requested gear through the gearbox linkage. In this case, high collision speed must be avoided. Accordingly, a larger cross-section area adjacent to the counter-side chamber is beneficial to maximize the force, which helps to slow down the piston. To reach Neutral, simultaneous loading of the working chambers is the commonly used strategy, in which case, asymmetric piston areas are the keys of the gear shifts. The floating piston, with its movement limited to one side of the actuator, assures the required area ratio for all three position-switches.

The system presented needs to be able to switch between the three dedicated positions (L, H, N), leading to six different switch tasks. The requirements and constraints of these are slightly different. For the comparison of the machine learning approaches, this article presents the case of Low-to-Neutral gear change. During this task, the controller has to meet the following qualitative requirements:

- The shift time must be less than 80 ms. As part of an AMT, the full change process consists of the clutch's disengagement, synchronization, actual gear change, and engagement of the clutch, from which the current task must not exceed the given limit to ensure minimal break in the power-flow of the drivetrain and ensure passenger comfort.
- The used solenoid valve switches must be less than six, per gear-change, to meet the product's lifetime requirements. Naturally, this restriction much more concerned with the average number of switches. Though in this way, it makes it impossible to use PWM like control strategies.
- The overshoot must be less than 1mm, to prevent unwanted gear shifts, which is critical regarding safety.

A. MODELING

The development of a nonlinear model of the actuator is presented in [33], which proved to be a useful environment for testing linear time invariant (LTI) controllers. The model is lumped since there are no spatial variations taken into consideration. Hence, the conservation equations are written as ordinary differential equations. Since the training of the reinforcement learning agents presented in this article requires an exceptionally high number of simulations, the nonlinear model needs to be further simplified.

The system model has two main parts: the solenoid valves and the actuator. The solenoid valve models are separated into electrical, dynamic, and airflow models. As the electric layout and most of the magnetic properties of the valves are unknown, lookup tables help determine the magnetic force, based on the time-series of the solenoid valve commands. To open the valve, the magnetic force has to overcome the viscous friction and the return spring. Armature position determines the cross-section area of the valve, from which the mass flow rate is calculated based on Bernoulli's equation for compressible fluids:

$$\frac{dm_{ch}}{dt} = \alpha_{fl} A_{fl} p_1 \sqrt{\frac{2\kappa_a}{\kappa_a - 1}\frac{1}{R_a T_1}\left(\pi^{\frac{2}{\kappa_a}} - \pi^{\frac{\kappa_a+1}{\kappa_a}}\right)}, \qquad (1)$$

where m_{ch} is the mass of air in the chamber, α_{fl} is the contraction coefficient, A_{fl} is the minimum cross section area, κ_a is the heat capacity ratio of air, R_a is the gas constant for air, T_1 is the source side temperature, p_1 is the source side pressure and π is the pressure ratio, calculated as:

$$\pi = \begin{cases} \frac{p_2}{p_1}, & \text{if } \frac{p_2}{p_1} \geq \pi_{crit} \\ \pi_{crit}, & \text{if } \frac{p_2}{p_1} < \pi_{crit}, \end{cases} \qquad (2)$$

where p_2 is the counter side pressure and π_{crit} is the critical pressure ratio.

The cylinder model contains thermodynamic models of the chambers and dynamic models of the pistons. The thermodynamic models determine the pressure, temperature, volume, and mass of air for each chamber. Their inputs are the mass flow rates of the valves, piston position, and the temperatures of the flowing air, while their outputs are the chamber pressures and temperatures.

Chamber volumes are calculated as a sum of the corresponding cylindrical areas with their height given by the position of the piston. The mass of air in the chambers is also known by integrating its mass flow rate and assuming ambient conditions at the start of the simulation. Air mass in the control chamber is also determined by (1), though as it exhausts to the environment, its minimum cross-section area is constant.

The pressure gradient is derived from the conservation equation of energy. For a given balance volume with p input and q output flows, the balance equation for the total energy is written as:

$$\frac{dE}{dt} = \sum_{j=0}^{p} \frac{dm_j^{in}}{dt}(h + e_k + e_p) - \sum_{k=0}^{q} \frac{dm_k^{out}}{dt}(h + e_k + e_p) + W + Q \tag{3}$$

where $\dot{m}_j^{in}$ is the jth input flow, $\dot{m}_k^{out}$ is the kth output flow, h, e_k and e_p are the mass specific enthalpy, kinetic energy and potential energy terms, W is the work term and Q is the heat transfer. The potential and kinetic energy terms are neglected, hence only the internal energy of the gas is taken into account. Therefore, the conservation equation is simplified:

$$\frac{dU_{ch}}{dt} = \frac{dm_{ch}}{dt}h - W_{ch} - Q_{ch} \tag{4}$$

where U_{ch} is the internal energy of the gas, and $\dot{m}_{ch}$ is the mass flow rate of the chamber. If the terms in the right side of (4) are extended, the equation takes the following form:

$$\frac{dU_{ch}}{dt} = \frac{dm_{ch}}{dt}c_pT_{inw} - p_{ch}\frac{dV_{ch}}{dt} - k_{ht}A_{ht}(T_{ch} - T_{amb}) \tag{5}$$

where c_p is the specific heat for constant pressure, T_{inw} is the temperature of the flowing air, V_{ch}, p_{ch} and T_{ch} are the volume, pressure and temperature of the chamber, k_{ht} and A_{ht} are the heat transfer coefficient and heat transfer area and T_{amb} is the ambient temperature.

By definition, the internal energy of an ideal gas is the following:

$$U_{ch} = c_v m_{ch} T_{ch} \tag{6}$$

where c_v is the specific heat for a constant volume. Correspondingly, the change of internal energy is also written as:

$$\frac{dU_{ch}}{dt} = \frac{d(c_v m_{ch} T_{ch})}{dt} \tag{7}$$

$$\frac{dU_{ch}}{dt} = \frac{c_v}{R_a}p_{ch}\frac{dV_{ch}}{dt} + \frac{c_v}{R_a}V_{ch}\frac{dp_{ch}}{dt} \tag{8}$$

By combining (4) and (8) the pressure gradient is expressed as:

$$\frac{dp_{ch}}{dt} = \frac{\kappa_a R_a T_{inw}\frac{dm_{ch}}{dt} +}{V_{ch}} \frac{-k_{ht}A_{ht}(T_{ch} - T_{amb}) - \kappa_a p_{ch}\frac{dV_{ch}}{dt}}{V_{ch}} \tag{9}$$

The chamber pressure is known, assuming ambient pressure as an initial condition. Once pressure, mass, and volume are determined, the temperature comes from the combined gas law. The mechanical models of the pistons calculate their state of motions, based on Newton's second law:

$$\frac{dv_{piston}}{dt} = \frac{\sum F_{pressure} - F_{friction} + F_{detent}}{m_{piston}}, \tag{10}$$

where v_{piston} is the piston's velocity, $\sum F_{pressure}$ is the sum of the pressure forces, $F_{friction}$ is the friction between the piston and the housing, which contains the Coulomb-friction and a viscous term, F_{detent} is the force of the detent mechanism and m_{piston} is the mass of the piston.

A sigmoid-function approximates the switching characteristic of the Coulomb friction, written as:

$$F_{friction} = F_{\mu}\left(\frac{2}{1 + e^{-v_{piston}f}} - 1\right) + d_{viscous}v_{piston} \tag{11}$$

where F_{μ} is the Coulomb friction, $d_{viscous}$ is the viscous friction coefficient, and f is the gradient of the sigmoid function.

In (10), the contact forces between the piston and the housing and between the pistons are neglected, though their effects must taken into account. The pistons' positions are saturated at their extremes, while between the two, the following assumption is used: if the main piston is between Neutral and Low, the floating piston stays at Neutral, otherwise they move together as one, rigid body. Therefore, the main piston's mass is calculated as:

$$m_{MP} = \begin{cases} m_{MP}, & \text{if } x_{MP} < 0 \\ m_{MP} + m_{FP}, & \text{otherwise} \end{cases} \tag{12}$$

where x_{MP} is the main piston's position, m_{MP} and m_{FP} are the main piston's mass and the floating piston's mass.

The floating piston's position is written as:

$$x_{FP} = \begin{cases} 0, & \text{if } x_{MP} < 0 \\ x_{MP}, & \text{otherwise} \end{cases} \tag{13}$$

The nonlinear model - with the contact forces modeled - has been verified, then it has been validated against measurements. Its accuracy is over 95% for the validated outputs, which are the chamber pressures and the Main Piston position. During validation, the model's uncertain parameters, such as contraction coefficients and friction coefficients, have been tuned to minimize the root mean square (RMS) error of the validated outputs. Fig. 2 presents the High to Neutral gear change as an example, while [34] presents the individual steps of the process and its results in details.

The first part of the research compares different machine learning algorithms, in which case a position-dependent sample time is used to minimize the time requirement of the learning, but also guarantee the model's stability around Neutral. After the conceptualization phase, the chosen agent's training has been repeated, but with a constant 0.01ms time step to model the real system's behavior with high accuracy.

B. TRAINING ENVIRONMENT

Discrete Partially Observable Markov Decision Process (POMDP) is chosen as a modeling paradigm. To interact with the agent, the set of observable states, the set of discrete actions, the state transition function, and the reward function need to be detailed. The action space, e.g., the control inputs of the system consist of the binary control commands of the two solenoids, leading to a four element discrete choice:

$$A = (0, 1, 2, 3) = ([0; 0], [1; 0], [0, 1], [1, 1]), \tag{14}$$

where the $[s_1; s_2]$ notation refers to the commands of the solenoids 1 and 2, respectively.

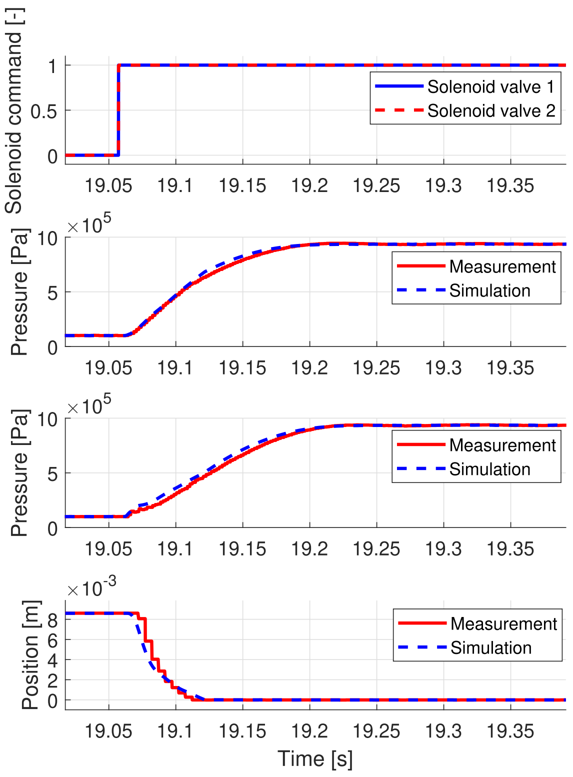

FIGURE 2. **Model validation - high to neutral gear change.**

The observable state space consists of the pressures and temperatures of the two chambers, and also the position and speed of the main piston, and finally, the supply pressure:

$$S = \left(p_{ch,1}, p_{ch,2}, T_{ch,1}, T_{ch,2}, x_{MP}, v_{MP}, p_{sup}\right) \tag{15}$$

The model in Section II-A describes the state transitions, with all the necessary hidden states, that are not reachable by the agent. An episode, e.g., a simulated control sequence can end in two ways. The first is the preferred ending when the main piston reaches its requested position, and its absolute speed falls under a certain margin. The second terminating condition is when the agent fails to fulfill its task, which occurs when the simulation exceeds the time limit, or the overshoot is above the predefined threshold, in this latter case, zero reward is given to the agent.

The rewarding system uses episodic rewards based on the encountered terminal states because there is no existing heuristic that assesses precisely every intermediate state of the episode. The terminal state is assessed based on the quality indicators of the performed actuation, which are the required time and the measure of overshoot. The part of the reward function that assesses the actuation time is formulated as follows:

$$R_T = 1 - \frac{t_s}{T_s} \tag{16}$$

where R_T is the sub reward for the duration of the actuation, t_s is the current actuation time, and T_s is the maximum provided time for the gear shift.

The overshoot sub reward is more complex, and it is formalized as follows:

$$R_O = \begin{cases} 1, & \text{if } x_{mp}^{max} < 0.5 x_{lim} \\ 2\left(1 - \dfrac{x_{mp}^{max}}{x_{lim}}\right), & \text{otherwise} \end{cases} \tag{17}$$

The final reward is the linear combination of the introduced sub rewards, with the weights (α_2, α_3) and a constant base reward (α_1) that is provided if the actuation is successful:

$$R = \alpha_1 + \alpha_2 R_T + \alpha_3 R_O, \quad \text{having} \tag{18}$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{19}$$

The requirements also limit the number of solenoid valve switches, hence it would be evident to count it as a terminal state and include it in the reward function. However, increasing the number of valve switches will eventually slow down the gear change, while short valve duties have no significant effect on the system's behavior. Thus, maximizing the actuation time and overshoot sub rewards will guarantee a low number of valve switches.

III. METHODOLOGY

As mentioned before, several techniques are used for controller design. Monte-Carlo Tree Search as a tree search algorithm (section III-A), supervised learning based agent on generated control data (section III-B), and various reinforcement learning agents, such as Deep Double Q Network (DDQN) (section III-D) and Policy Gradient (section III-E) are developed. To further enhance the performance, mixed solutions, such as the Planning agent (section III-F), and a new, PG-MCTS algorithm (section III-G) is also designed. The utilization of these six methods gives a good comparison of machine learning possibilities for such tasks.

Fig.3 shows the basic control loop used for this research. Translating to classic control terms, the environment serves as the system or plant to be controlled, and the agent plays the role of the controller. In the training phase, a simulated model of the actuator is used, since it is faster, does not wear the plant, and the implementation of the learning algorithms on the actual ECU is not feasible. The model takes the two solenoid commands as control input described in (14) and generates the state vector represented in (15). The controller architecture differs for the methods presented. The simplest controller uses only the neural network, which translates the state vector to the action at each step. This is utilized in the DDQN, PG, and SL algorithms and the Planning agent's prediction phase (actual control). The MTCS uses a model-based search for the controller. The tree search is combined with the neural network for the Planning agent's training and the prediction of the PG-MCTS. Because of the nature of the applied methods, there are no human-designed or handcrafted strategies, but the learning processes develop the control logic through search, or trial and error.

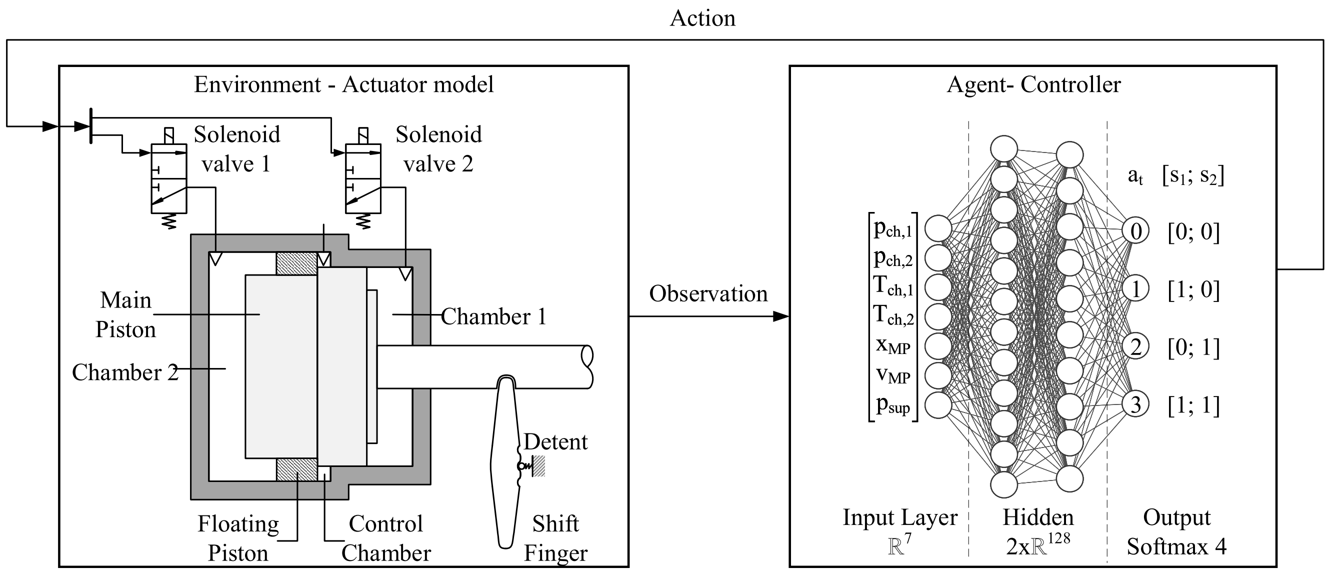

FIGURE 3. **Interaction between the agent and the environment in reinforcement learning.**

A. MONTE-CARLO TREE SEARCH

Monte-Carlo Tree Search is a best-first search algorithm that constructs a tree-based representation of a problem incrementally by repeating the following four steps:

- Selection: Recursively selects the nodes with the highest "Upper Confidence bound applied for Trees" (UCT) value, until the leaf node is encountered.
- Expansion: Populates the child nodes with the generative model of the environment, if there is any.
- Simulation: Carries out an MC rollout until the end of the game.
- Backpropagation: Updates all the parameters in the selection path based on the result of the simulation.

MCTS earned the researchers' attention thanks to its results in the challenging domain of Computer Go [35], [36]. The main contribution in MCTS is the so-called Upper Confidence bound applied for Trees algorithm that considers every node selection routine as a multi-armed bandit problem. Hence a bandit-algorithm determines the pathway from the root of the tree to its leaf. The UCT is formed as:

$$\overline{X_i} + 2C_p\sqrt{\frac{2\ln N_i}{n_i}} \tag{20}$$

where $\overline{X_i}$ is the average value of the given node, C_p is a constant that provides an additional way of controlling the exploitation-exploration trade-off, N_i is the number of visits of the ancestor node while n_i is the number of visits of the given node. For more details see [37]. MCTS converges towards the globally-optimal solution, though it also has some shortcomings in such domains, as it can not operate in a real-time manner, because of the high number of the required planning iterations for a single decision. However, it still can be utilized as a benchmark in simulation for other methods; as a training sample generator for supervised learning; or as a meta-heuristic for hybrid methods where the resulting system can function in real-time. Fig. 4 presents the tree built by MCTS for a single decision. To better represent the populated, visited, and optimal solution, a larger sample time ($3ms$) was chosen.

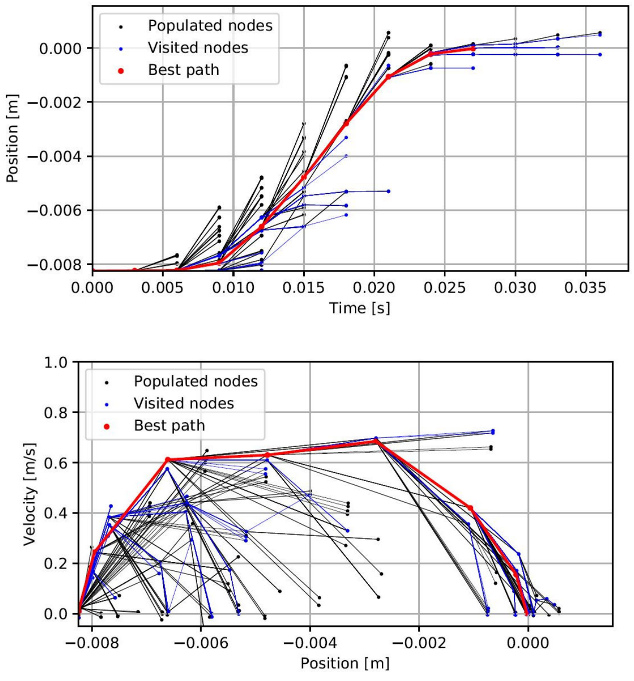

FIGURE 4. **Example of the MCTS algorithm at a sample time of 3*ms*.**

B. SUPERVISED LEARNING

The goal in supervised learning is to map input vector x to output label $y = f(\theta, x)$, where θ contains the weights of a neural network. Hence, SL can be a conspicuous choice for control problems with such a low sample time because only the training phase is time-consuming, while the forward pass function of the neural network is easily applicable in real-time. Despite the impressive results in several areas [38], this method has some practical drawbacks. The most concerning is the enormous need for labeled training data, whose generation or collection is very resource-intensive or, in some cases, impossible. Moreover, the reachable solution set is sometimes bounded by human knowledge, because it is only capable of resolving a previously fixed problem, i.e., reestablishing an existing connection between the input and the output. In the current case, these difficulties are resolved by using the MCTS algorithm for generating the training samples. Table 1 summarizes the final setup used for the training of the supervised learning agent.

TABLE 1. Training setup for supervised learning.

Parameter	Value
Learning rate (α)	0.00001
Num. of hidden layers	2
Num. of neurons	128,128
Layers	Dense
Hidden layers activation function	RELU
Number of training samples	80000
Epochs	500
Batch size	256
Number of folds	10
Optimizer	Adam
kernel initializer	Xavier normal

C. REINFORCEMENT LEARNING

In several domains, there is no such opportunity to generate sufficient training samples that represent a considerably better solution in terms of performance and quality. In these cases, reinforcement learning can be utilized since an agent's capabilities are not bounded by the concepts and schemes or solutions of human experts. This objective is reached through the trial and error based approach that determines the interactions between the agent and the environment, where the only compass that an agent has is a scalar value from the environment called reward. Consequently, the experiences used for training are obtained in an online manner over time. The agent endeavor is to develop a behavior that helps reach its goals, hence maximizing the cumulative reward called return:

$$G = \sum_{t=1}^{T} \gamma^t r_t, \tag{21}$$

where γ is the so-called discount factor specifying how the decisions of the present affect the reward of the future, and r_t is the reward obtained at time step t. Fig. 3 shows the basic training loop of reinforcement learning. Nevertheless, RL also has some weaknesses in terms of robustness, reliability, and reproducibility [39]. Along with the most important value-based and policy-based learning methods, a simplified version of Deepmind's AlphaGo Zero architecture is also implemented, which seriously mitigates the performance concerns of RL by using MCTS in the policy iteration procedure as a policy improvement and a policy evaluation operator. Moreover, a novel version of the UCT algorithm is introduced, which enables us to enhance the performance of the trained neural network in prediction, while real-time applicability is maintained.

D. VALUE-BASED METHODS

Deep Q-learning is a popular form of value-based RL, since the breakthroughs of Deepmind's researches in several areas [40], [41]. In this concept, the function approximators θ parameters are tuned to predict the action-value function in every state over the executable moves, for that the Bellman equation is used as an update rule:

$$Q(s_t, a_t; \theta_t) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t; \theta_t^-). \tag{22}$$

where $Q(s, a)$ is the action-value of the action a in state s, and s' is the resulted state of the state transition of the environment triggered by the execution of the chosen action a in state s. θ_t is the weights of the online neural network, and θ_t^- is the weights of the target network in time step t. The predicted values in a given state characterize what to expect from each choice in the long run. Hence these values need further interpretation to formulate a policy, which is done by the agent. For this matter, Deep Double Q-learning is chosen from the realm of methods associated with Deep Q-learning. The DDQN is preferred over the original DQN because it is more stable and learns more robust policies thanks to the decoupled selection process in the formulation of the targets for training. The update rule looks slightly different:

$$\begin{aligned} Q(s_t&, a_t; \theta_t) \\ &= r_{t+1} + \gamma Q(s_{t+1}, \underset{a}{argmax} Q(s_{t+1}, a_t; \theta_t); \theta_t^-). \end{aligned} \tag{23}$$

The decoupling concept is shown by (23), where the role of the online neural networks is to evaluate the action over the argmax function, while the target network estimates the value of the chosen action.This approach helps to eliminate the positive bias from the predicted values caused by the maximum overestimated values used as an estimate of maximum value [42]. More details about the algorithm is given in [43]. The training setup for the DDQN algorithm is described in Table 2.

E. POLICY-BASED METHODS

Policy-based RL has become an interesting realm of model-free algorithms, thanks to the results in several challenging domains [44], [45]. In this concept, the policy is approximated directly, hence the function approximators θ parameters are tuned to predict a probability distribution in

TABLE 2. Training setup for deep double q-learning.

Parameter	Value
Learning rate (α)	0.0001
Discount factor (γ)	0.99
Batch size	512
Experience memory size	15000
Num. of hidden layers	3
Num. of neurons	256,256,256
Hidden layers activation function	RELU
Method	DDQN
Freq. of weight sharing (in ep.)	100
Layers	Dense
Optimizer	Adam
kernel initializer	Xavier normal

every state over the executable moves. Therefore it operates as a dynamic heuristic because the meaning of the predicted values is only comparable for the particular branch. The advantages of this method over the value-based ones are guaranteed convergence toward a local optimum and improved stability, which can be further enhanced by using different parameter update frequencies. The parameter update rule is formalized as follows based on [46], [47]:

$$\theta \leftarrow \theta + \alpha \nabla \log \pi_\theta(s_t, a_t) \sum_{t=1}^{T} \gamma^t r_t \tag{24}$$

where $\pi_\theta(s_t, a_t)$ is the choice probability of action a in state s at time step t predicted with the neural network's θ parameters, and α is the learning rate. The training setup for the Policy Gradient algorithm is described in Table 3. Accordingly, the PG agent operates as follows:

TABLE 3. Training setup for policy gradient.

Parameter	Value
Learning rate (α)	0.0001
Discount factor (γ)	0.99
Num. of ep. after params are upd. (ξ)	10
Num. of hidden layers	3
Num. of neurons	256,256,256
Hidden layers activation function	RELU
Layers	Dense
Optimizer	Adam
kernel initializer	Xavier normal

1) Initialize the neural network's θ parameters according to the Xavier-normal kernel initializer scheme, along with the initial state s_0.
2) Carry out interactions with the environment until a terminal state is encountered, then store the history of the episode.
3) If the update frequency ξ is reached, then the rewards gathered into the history have to be discounted.
4) Finally, the gradients are calculated based on the collected experiences, and the neural network's θ parameters are updated accordingly.

F. THE PLANNING AGENT ARCHITECTURE

This architecture implements a simplified version of Google's AlphaGo Zero [48] for this control task, to enhance all the performance indicators of RL algorithms that are inferior. Simple model-free RL algorithms often seem to sample inefficient and get stuck in an unsatisfactory performance level in many domains. Consequently, different enhancement methods are integrated into RL algorithms for boosting performance. This robust architecture is associated with a unique two-headed neural network construction, where one of the heads approximates the state-value function, while the other the choice probabilities, hence the policy directly. These two heads are utilized inside the MCTS. The main idea of the concept is the synergy between the RL and MCTS, which results in a neural-network controlled tree search algorithm, where the prediction of the value-head is used as a fast rollout strategy instead of MC rollout, in consequence, it manages the exploitation. At the same time, the policy head regulates the exploration with the choice probability of the given node. A variant of the polynomial upper confidence trees (PUCT) algorithm is used for node selection:

$$Q(s, a) + P(s, a) C \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{25}$$

where $P(s, a)$ is the choice probability of the action a in state s, C is a constant that provides an additional way for controlling the exploration-exploitation trade-off, $\sum_b N(s, b)$ is the sum of visit counts of the given branch, and $N(s, a)$ is the visit count of the given node. MCTS evaluates the given state over a horizon, which limited by the number of iterations. Moreover, it outputs a probability distribution based on the visit counts of the edges branching from the given state. In the training loop, the action is chosen based on the mentioned probability distribution according to the "robust child" method [49], hence the one with the highest probability and the neural network is tuned with a particular loss function:

$$l = (z - v)^2 + \pi^T \log p \tag{26}$$

where z is the final reward, v is the value of a state predicted with the value-head, π^T is the probability distribution created with the MCTS, and p is the probability distribution predicted with policy-head of the neural network. This expression incorporates the loss of the two-heads, by using the mentioned probability distribution for training the policy-head, while the value-head is trained with the final reward of the given episode. For more details [48]. The training setup for the Planning agent is shown in Table 4.

G. PG-MCTS

AlphaGo architecture enhances the performance of an agent through the integration of planning into learning, but it can also be utilized in the prediction phase. Unfortunately, it requires a serious amount of planning to reach considerably better results. Thus real-time applicability can not be maintained on an enhanced performance level. The reason

TABLE 4. Training setup for the planning agent.

Parameter	Value
Learning rate (α)	0.0001
Discount factor (γ)	0.99
Batch size	1024
Experience memory size	50000
Num. of hidden layers	2
Num. of neurons	256,256
Hidden layers activation function	RELU
Layers	Dense
Optimizer	Adam
kernel initializer	Xavier normal

is, it does not exploit any domain-specific knowledge, which could decrease the required planning time and meaningfully improve performance.

Domain-specific knowledge can be interpreted as insights built into the search tree in the form of cut-offs. Still, this approach would turn the results to be "handcrafted". The UCT algorithm is modified in this research to avoid such outcomes and exploit specific features of the control task, resulting in more efficient exploitation of the trained neural network's hidden expertise.

In this environment, the very first step determines the appropriate control strategy for the particular episode, thus the reachable final reward, but still actuation time also has a significant role in the calculation of the reward. Consequently, a single inappropriate decision can not stop the piston from reaching the Neutral, it just extends the required time, while the last few actions are for fine-tuning, because the top of the line results are very close to each other most of them are on the same branch or neighboring branches of the tree. In consequence, a narrow and deep tree has to be built, which spreads in the last few layers. Since the algorithm is used in the prediction phase, firstly, the type of the trained agent has to be chosen, which performance can be enhanced. An agent which approximates the action-value function is not the best choice, because in practice the predicted values are very close to each other, which is not a problem if the decision is made with an argmax function, but in the UCT algorithm, it results in a tree search which spreads as a breadth-first search algorithm.

In contrast, an agent that directly approximates the policy naturally distinguishes better between actions thanks to its dynamic heuristic like operation, but it also has its drawbacks. Dynamic heuristic like operation means that the predicted values do not carry information about the efficiency of the long run; it only reflects the challenges of the given scenario. Accordingly, a high choice probability not necessarily means a great outcome. This behavior easily blindsides the algorithm through the backpropagation procedure, which results in wasted planning time. The proposed formulation of the UCT algorithm overrules this behavior by using the predecessor choice probability to calculate the given edge choice probability:

$$P_i^p P_i + C\sqrt{\frac{\ln N_i}{n_i}}, \tag{27}$$

where P_i^p is the policy output (choice probability) of the parent node and P_i is the policy output of the child node.

Thanks to this modification in the first few layers of the tree, the exploration can not waste iterations by overruling the exploitation, but in the last few layers, it starts to spread with the guidance of the backpropagated real values of terminal states attached to leaf nodes, because the horizon of the planning reaches terminal states. This approach also exploits the phenomenon that in the first few layers, only one choice has a high choice probability, and as the iterations reach deeper layers in the tree, the predicted probability distribution that provides the choice probabilities starts to flatten.

The search tree built by the MCTS algorithm presented in Fig. 4 needs extensive planning to find the right track and perform the first step, which makes real-time applicability impossible. In contrast, Fig. 5 shows an example of the PG-MCTS algorithm, which only plans a few steps forward in each move by using a policy network. It results in a performance increase, as shown in Section IV, while it needs considerably less resource than the MCTS since the tree for the entire solution has fewer nodes than in the case of pure MCTS algorithm which is built for one step only.

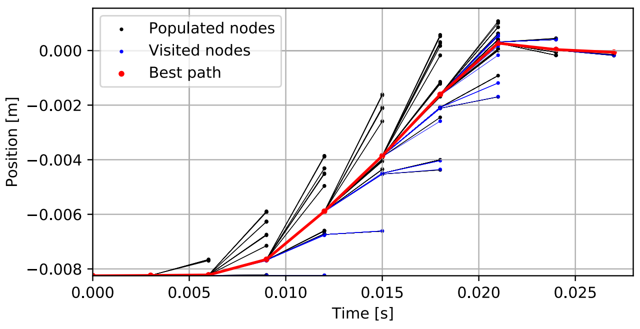

FIGURE 5. Example of the PG-MCTS algorithm at a sample time of 3*ms*.

IV. SIMULATION RESULTS

This section presents further training details and the performance and strategy choice comparison of the different methods. This requires that all data for the performance figures are generated by applying all methods for the same seed of environmental parameters, for 10000 simulations, to ensure representativity and to enable in-depth comparison.

Data preparation and hyperparameter optimization are crucial parts of supervised learning; hence, the training samples generated by the MCTS algorithm are normalized, filtered, and shuffled. Normalization of every input vector is essential since it makes the optimization problem better conditioned. Filtering is also necessary because the raw training set may contain some degenerated samples, which ultimately deteriorates the neural network's accuracy. In this particular case, the MCTS algorithm produces such samples, because, in some episodes, there are actions that have delayed effect on the observable state in the first few steps, while MCTS plays a mixed strategy thanks to the MC rollouts and the bandit based operation of the UCT. Thus the chosen action could vary even if the initial state does not change. Consequently, all the samples that show this behavior have to be removed to reach the highest possible accuracy. The data generation phase is advised to collect an equal number of samples from every class to prevent over and underrepresentation. Shuffling is also critical because the same outcome has to be reached in every batch fed into the neural network. Thus the batch size also has to be chosen accordingly. The other hyperparameters are chosen via trial and error. The performance of the supervised learning agent and the original MCTS are shown in Fig. 6. The figure shows the distribution of the earned rewards during the evaluation.

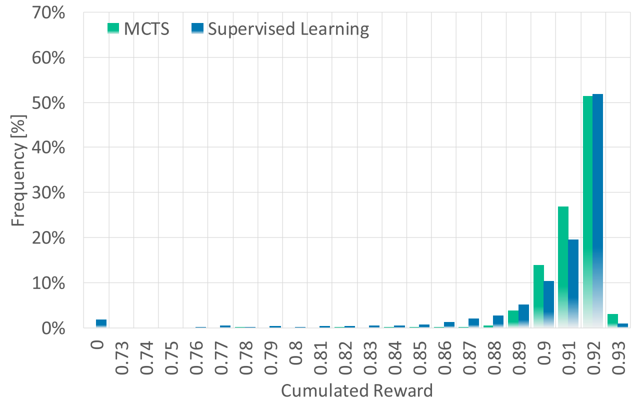

FIGURE 6. Comparison of the MCTS and SL algorithms, based on the distribution of the earned rewards.

It is clear that the two algorithms have the same behavior, and they produce the same characteristics, which is expected, since the MCTS generated the training data for the SL agent. Unfortunately, the trained neural network has a 1.89% failure rate, while the MCTS algorithm solves every episode and has a higher average score. Though, the operation speed of the neural network is way faster than the MCTS algorithm as it is expected.

Fig. 7 shows the convergence of each algorithm. As expected, the PG algorithm converges faster than the DDQN and seems more stable too. Still, the best one is the Planning agent which has an absolute superiority by any measure. RL is addressed as the field of ML that is not biased by human knowledge, though it is inevitable in the formulation of the reward function and credit assignment. Since the reward function does not alter in any solution, the difference in credit assignment schemes triggers the discrepancy between the algorithms. Credit assignment is one of the most complex challenges of RL. The ideal solution would be a reward strategy that assesses every step in the episode based on its role in the entire process. Still, rewards strategies are more like heuristics, and they come with no guarantees. The opposite is a final reward, which is discounted to assess all the moves in the episode. Unfortunately, the discounting concept assumes particular causation between slices of the episode, which results in an inappropriate assessment of individual

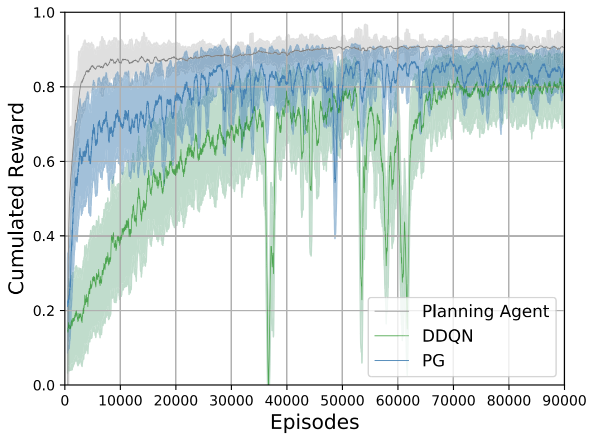

FIGURE 7. Convergence of the planning agent, DDQN, policy gradient algorithms, based on the distribution of the earned rewards.

steps. Although these issues are supposed to straighten out over the training, they always reached deficiently.

In contrast, the Planning agent architecture can assign a reward for every move, that is validated over a specific horizon with MCTS. Consequently, this approach assesses the individual steps more reliably than the discounting concept, and it results in improved convergence properties and performance. However, a neural network drives the MCTS. Thus, it can end up in failed episodes, but there are no erroneously promoted moves even in these cases. Thanks to the incorporated loss function, the value-based part operates as a regulator for such scenarios by using the final reward of the given episode, which can be considered a valid measure. This approach can be interpreted as an endeavor toward an area of RL that entirely lacks handcrafted knowledge and schemes.

Fig. 8 shows the comparison between the DDQN and PG algorithms. The PG algorithm has a better average score and a lower share in failed episodes, which is only 0.22%, but it still can not settle every try. DDQN fails only 1.1% of its tasks, which is lower than the previously introduced SL's, despite that SL has a higher average score. This comparison shows

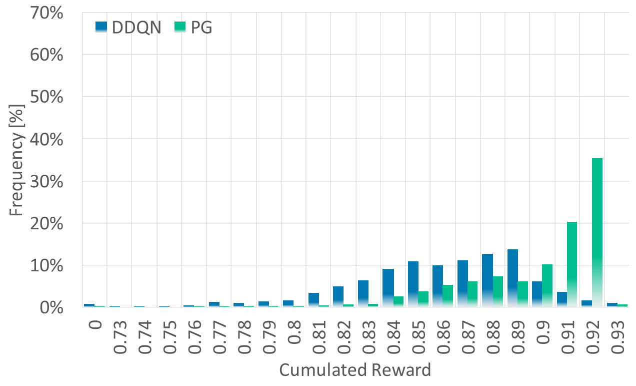

FIGURE 8. Comparison of the DDQN and PG algorithms, based on the distribution of the earned rewards.

that simple ML-based methods are not able to outperform the pure MCTS algorithm expect in their real-time capabilities.

The PG-MCTS and the Planning agent represent the best performance, although the PG-MCTS reaches a higher average score by further increasing the frequency of the best possible solutions. Moreover, these algorithms are also capable of eliminating all the failed episodes as pure MCTS, and in the meantime, these methods maintain real-time applicability. The comparison of Fig. 6 and Fig. 9 show that these algorithms produce the same characteristics, but with higher spikes in the section of the best reachable solutions, the PG-MCTS algorithm solves more than 70% of the episodes with the two best achievable solutions. It is interesting, though, that not the MCTS has the best average score, the reason is that the planning time has to be scaled back because of limited resources.

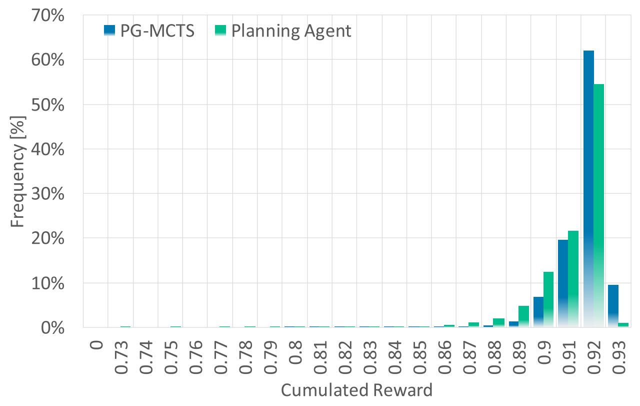

FIGURE 9. Comparison of the PG-MCTS and the Planning agent algorithms, based on the distribution of the earned rewards.

A. STRATEGY CHOICE COMPARISON

After the precise assessment and comparison of each algorithm, it is interesting to understand the strategic wise differences beyond the performances. Fig. 10 shows the applied strategies on an abstract level by the frequency of each action used, to be representative, every algorithm used the same seeds as in the performance comparison case. Apparent similarities can be observed between the SL and MCTS algorithms, which confirms the suggestion that they have the same behavior, and the training of the neural network is successful. The same phenomenon occurs between the PG-MCTS and the Planning agent, but, interestingly, there is no significant similarity between them and the MCTS, although they have nearly the same characteristics in the performance figures. The strategy of the DDQN and the PG does not show similarities with each other or any other method, but it is compelling that the PG entirely ignores one of the actions. It is fair to say that every method operates as a filter that enables specific behaviors to conglomerates into a unique strategy.

To gain further insight into the differences in strategy choice and performance between individual agents, it is advisable to compare their decisions across the state of space. This comparison is aided by Fig. 11, which shows

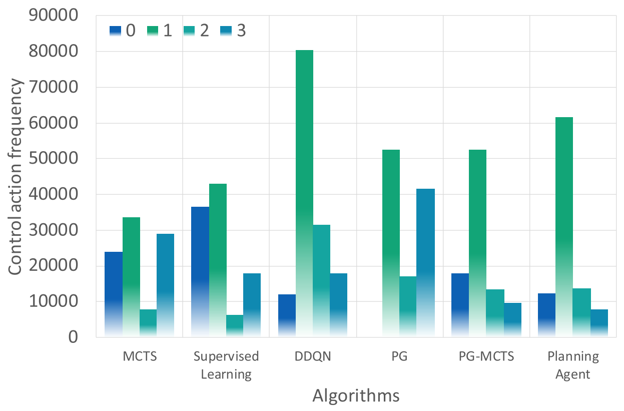

FIGURE 10. Comparison of strategies on an abstract level.

the chosen actions of the agents in latent space. Since the state vector has seven elements, the Principal Component Analysis (PCA) dimension reduction algorithm is used for visualization, which maps the $\mathbb{R}^7$ observation space to two dimension $\mathbb{R}^2$ latent space, which is simply a representation of compressed data in which similar data points are closer together in space. PCA aims to reduce the dimension of a data set of a large number of related variables while preserving the largest possible variance in the data. The PCA finds new variables as the linear combinations of the original ones [50]. Thus, by projecting the original parameter space into two dimensions, the distribution of different actions for different agents can be displayed. Since the new state-space parameters have no physical meaning, the labels are omitted in the representation in Fig. 11. This representation serves two purposes: The first is the comparison of the action choice of different agents, while the second is to determine whether the agents use mixed or clear strategies. Furthermore, for clarity of representation, only two class distributions are displayed: Class 1 (valve 1 opens, as red) and Class 3 (Both valves open, as blue).

The first question is why the MCTS agent reward values are not the best, as it converges to the optimal solution in principle. However, both the Planning agent and the PG-MCTS algorithm are superior. Naturally, the previous statement is only true if the MCTS planning number also converges to infinity [37]; or, in practice, is large enough to build a tree that covers the prediction horizon with high confidence. Intuition says that this would result in a clear strategy choice, which does not happen, as shown in Fig. 11a. MCTS does not segment the state space and relies heavily on random rollouts. However, as the end of the episode comes closer and the horizon converges, the algorithm finds an optimal solution for the rest of the task that explains the high rewards.

Though the SL agent is trained by the data generated by the MCTS and both their performance and action choice (Figs. 6 and 10) are similar, the SL needs to "regularize" the hybrid action choice during the training. The "regularization" is done by the neural network's generalization feature, which leads to a clearer segmentation of the state space,

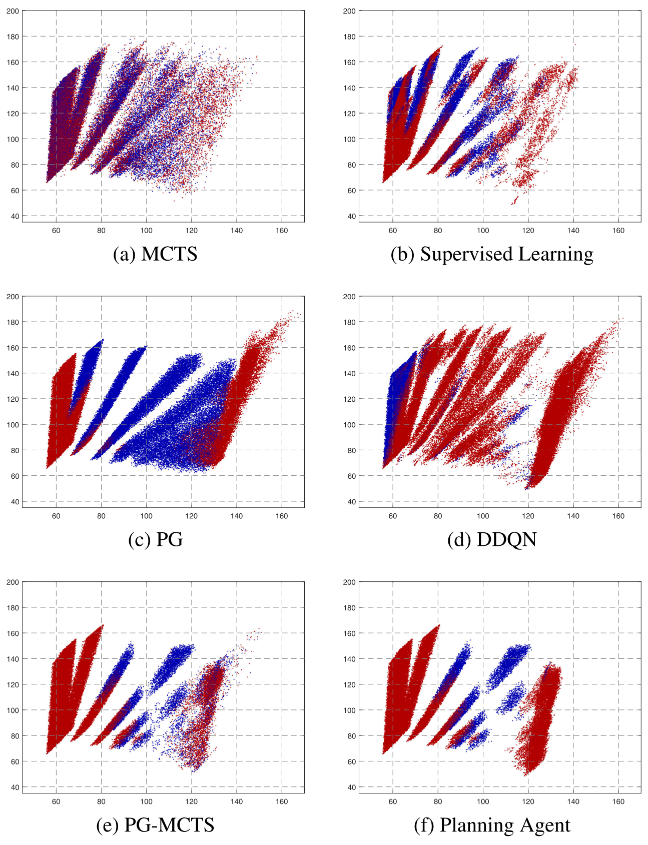

FIGURE 11. **Distribution of chosen actions (Class 1 - red; Class 3 - blue) on the latent space reduced by PCA.**

as shown by Fig. 11b. It is an interesting fact, which is shown in Fig. 11d, the DDQN agent's diagram differs the most from the others, which can be explained by the fact that it has no connections to any other examined methods. Also, the PG agent shows a quite simple classification diagram (Fig. 11c), which comes from the simplicity of the agent itself and gives a slight insight into why these simple agents fail in more complex problems.

Having some similarities in the actor network's paradigm, PG, PG-MCTS, and the Planning agent show similar classification diagrams, though in Fig. 11e, the "noisy" effect of the PG-MCTS's state space can be examined. The PG-MCTS uses an incredibly small amount of planning to maintain real-time applicability, which emphasizes the effect of the improper planning time on the strategy choice. The comparison of Fig. 11e and Fig. 11f provides an intuition about the effect of planning utilized in prediction and learning. The integration of planning into learning shows much more confidence in strategy choice, which is also accomplished by the generalization feature of the neural network, but still, PG-MCTS has superiority.

B. METHOD SELECTION FOR EXPERIMENTAL VALIDATION

In the previous sections, several methods have been presented for solving the same control problem. Based on the performance, the PG-MCTS algorithm would be the obvious choice, unfortunately, the controller's CPU does not have enough computational power to run the algorithm in real-time. The second best option is the Planning agent. For the real-word test, the agent can not be trained in the simplified environment, which means longer episodes and computationally more exhaustive planning in every single step, which could result in weeks for one training not to mention hyper-parameter optimization. The same goes for the pure MCTS algorithm, however, it can not operate in real-time, not even with the simplified environment. The DDQN algorithm has the lowest average score of all, so it can not be a reasonable choice. From the remaining algorithms, the PG is chosen because its average performance is higher than the SL's, while its share in failed episodes is only 0.22%, moreover the neural network trained with SL has more neurons because it can not be scaled back further without meaningful performance loss, hence it would require more memory which is also limited on the controller.

V. MEASUREMENT SYSTEM

Training the agent on the real target exceeds the system's capacity; hence, the agent is trained in a python simulation environment and is only validated in the embedded environment. Once the agent is trained, it is implemented in Matlab/Simulink to merge it with the given function software, and then the C-code is generated from the algorithm with Simulink Coder. At last, the C-code is compiled and downloaded to the ECU. Fig. 12 shows the testbench. It contains a 3-stage, 16-gear heavy-duty gearbox with two electric motors simulating the vehicle's engine and the road resistance, a commercial actuator control unit, a pressure sensor, and a high precision laser position sensor.

The inputs of the Neural Network are the chamber and supply pressures, the Main Piston's position and its velocity. In the industry, one important goal is to minimize the number of sensors required to control a system. In this way, the cost of the product also can be reduced. Therefore, the chamber pressures are estimated by a model-based observer, which uses the equations of the nonlinear model. A commercial pressure sensor measures the supply pressure with 0.02bar resolution, and a high precision laser sensor measures the position of the Main Piston. The piston's velocity is derived by the measured position, and then it is filtered.

The signals are measured via CAN with a Vector VN1630A measurement device [51], and are processed with Vector CANape 14 measurement software [52]. All signs are measured with 1ms sample time, which is also the frequency of the control algorithm.

The major challenge during implementation and testing is to develop an agent concerning the hardware limitations of a commercial ECU. In this case, the maximal CPU frequency is 160MHz, and the program memory is 2048kB. Still, a part of the resources is already reserved for other functions, most importantly to the automotive ECU framework that holds the safety-related algorithms.

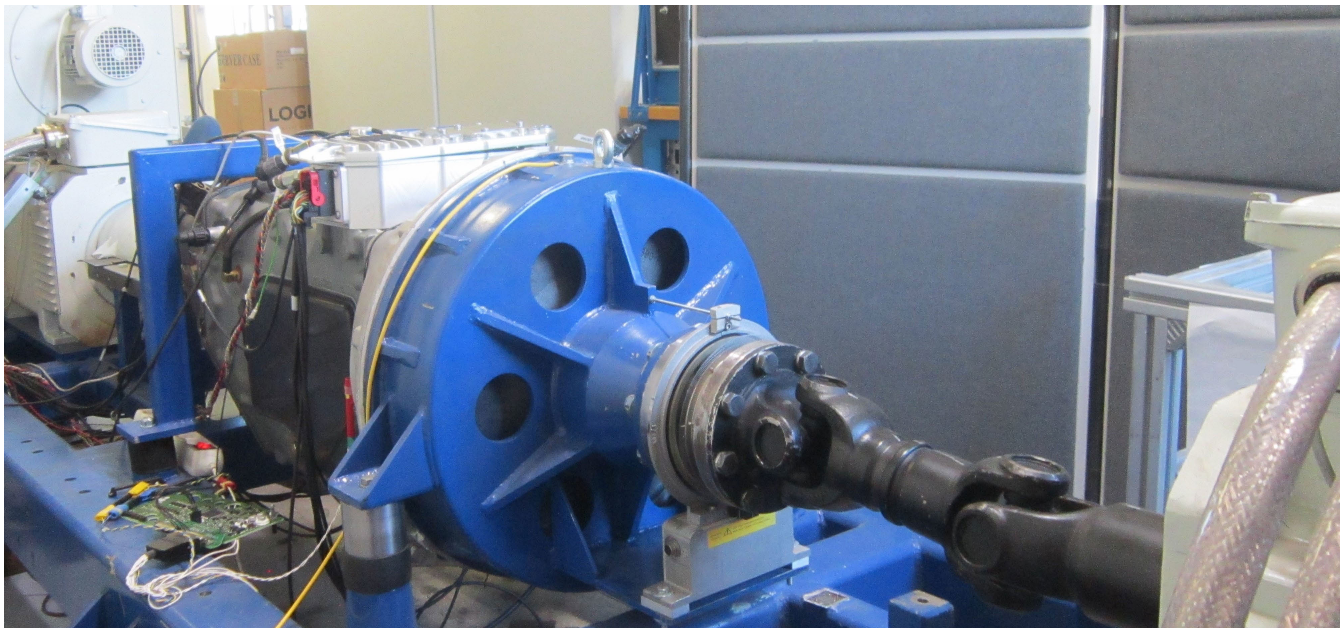

FIGURE 12. Measurement system.

By iterative testing and optimization, the size of the Neural Network has been reduced. The smallest Neural Network, which is suitable to control the system, contains four inputs and one hidden layer with 32 nodes. The inputs are divided into two parts: measured and calculated ones, as mentioned before. Prediction of a feedforward Neural Network with ReLU activation layers consists of linear algebraic operations and saturation. In this case, the network has a node number of (7, 32, 4) for the input, hidden, and output layers, respectively. This leads to $(7*32)+(32*4) = 352$ multiplications, 32 sums and RELU-s, and the 4 element softmax calculation. The CPU can manage these calculations in real-time. Hence, the bottleneck is the element size of the weight matrices. Naturally, training on the actual hardware would require much more resources, which is why it was omitted.

A. MEASUREMENT RESULTS

The gear change is performed between 4 bar and 10 bar supply pressure with 0.5 bar resolution. Fig. 13 shows three selected test cases, through which the applied strategy of the agent is presented. The first and second diagrams show solenoid valve 1 and solenoid valve 2 commands for 5 bar supply pressure, the third and fourth diagrams show the solenoid valve commands for 8 bar supply pressure and the fifth and sixth diagrams show the solenoid commands for 10 bar supply pressure. The seventh and eighth diagrams show the chamber pressures, and the ninth diagram shows the main piston's position for the three cases.

The piston is moved to Low position by direct solenoid valve commands, and then at $0s$, the supervisory logic passes the control to the agent, which achieves gear change. In the case of 10 bar supply pressure, the agent applies a typical open-loop control strategy: it loads both working chambers at the same time until Neutral is reached. While it is not the fastest possible method, it is a very robust and reliable control strategy. As the supply pressure is reduced to 8 bar, the piston's speed lowers, hence it takes more time to reach Neutral. This induces more extended solenoid valve commands, but with the reduced supply pressure, the risk of overshoot also decreases. If the supply pressure further decreases, the agent becomes "braver", thus it enables a higher pressure difference between the chambers, which results in faster gear changes despite the lower supply pressure.

On the contrary, the PG agent tested in the simulation first activates only solenoid valve 1 to move the piston towards Neutral, then it uses solenoid valve 2 to slow down the piston and prevent overshoot. This is a high-risk, high-reward strategy, as it has exceptional performance, but it requires perfect timing. Presumably, the key difference between the two agent lies in their inputs. The piston's velocity is removed from the input of the agent presented in this paper as it could only be calculated from the measured position and it would suffer from noise. However, without the piston's velocity, it is more difficult to predict overshoot as it a becomes partially observable Markov decision process, therefore to reduce the risk of failure, the agent learned different control strategies with respect to the supply pressure.

The measurements are summarized in Fig. 14. The first and second diagrams show the solenoid valve commands, the third and fourth diagrams show the chamber pressures, and the fifth diagram shows the main piston's position.

As expected, the extremes (4 bar and 10 bar supply pressure) envelop both of the chamber pressures: if the supply pressure is higher, the pressure gradient must be steeper.

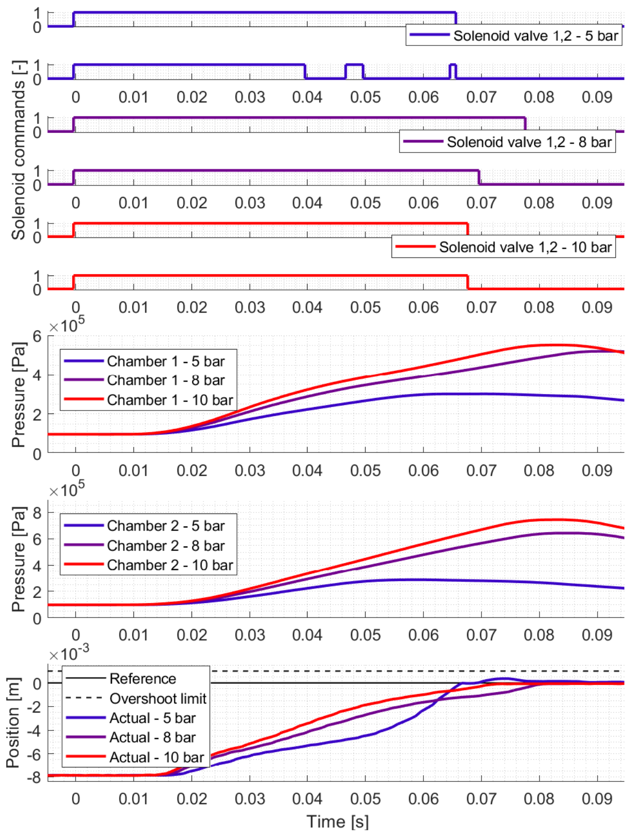

FIGURE 13. Measurement results on 5, 8 and 10 bar supply pressure.

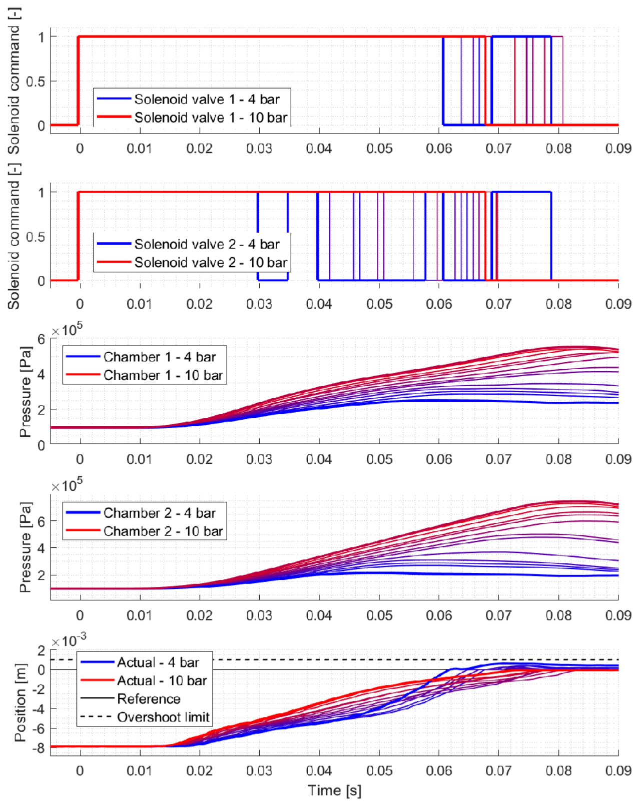

FIGURE 14. Summary of the measurement results.

TABLE 5. Performance of the PG agent on testbench.

Supply pressure [bar]	Shift time [ms]	Overshoot [mm]	No. of valve commands [-]
4.0	60	0.6	6
4.5	65	0.4	2
5.0	64	0.4	4
5.5	63	0.3	5
6.0	65	0.2	5
6.5	74	0.02	3
7.0	74	0.03	3
7.5	75	0.02	3
8.0	77	0	2
8.5	75	0	2
9.0	74	0	2
9.5	71	0	2
10.0	71	0	2

However, this is not the case with the solenoid valve commands and the piston's position. One would expect that the gear shift with the highest supply pressure would have the fastest gear change, hence the shortest solenoid valve commands and as the pressure decreases, the gear change should slow down, and the solenoid valve commands should become longer. However, the agent's goal is to maximize its reward while achieving successful gear change. A gear change fails due to two possible reasons: either the gear shift time exceeds $80ms$, or its overshoot is higher than the allowed $1mm$. In the case of higher supply pressure, the risk of exceeding the overshoot limit is greater than exceeding the maximum shift time, while in case of lower supply pressures, the risk of overshoot is lower. Hence, to minimize the chance of failure, the agent developed different strategies depending on the supply pressure. Between 10 bar and 8 bar supply pressure, the agent uses a conservative strategy, which is mostly used for open-loop control of these systems. Then, the control strategy starts to change under 8 bar supply pressure. This change becomes very prominent between 6 bar and 5.5 bar supply pressure as there is a huge difference in the position signals. This change can also be seen in the chamber pressures, but its effect is not as significant as on the piston position.

Table 5 summarizes the performance of the PG agent, which also confirms the conclusions made based on Fig. 13 and Fig. 14. On high supply pressure, the agent aims to prevent overshoot, hence it uses a conservative control strategy, which is very similar to open-loop strategies. It uses only two solenoid valve switches, which eliminates overshoot by loading the chambers while under 8 bar pressure, this strategy would exceed the available time. Therefore, on lower supply pressures, the strategy of the agent changes. It permits higher overshoot, hence achieves faster gear change, while using shorter, but more frequent valve switches in the counter-side chamber to control the piston's movement.

While in simulations, the PG agent failed in 0.22% of the episodes, it performed exceptionally well on the real target, as it successfully executed all test cases. In simulations, the environmental and supply pressures and also the ambient temperature are changed randomly in a wide range of scale, which can result in extremely difficult scenarios. Although, during the experimental tests, both ambient states remained around their standard values as only the supply pressure could be modified, resulting in simpler operating conditions.

B. COMPARISON WITH LTI METHODS

In [53] the development and testing of a gain-scheduled PID controller and an LQR controller are presented. As the developed controllers were tested on the same system, their results can be used as a benchmark to justify or contradict the PG algorithm choice. However, for exact comparison, the measurement results have been re-evaluated. In [53] four different test cases have been analyzed and the exact positions of the High and Low gears scatter around a nominal value, hence the shift time was calculated between the new request's emergence and reaching 90% of the reference position. This way, the performance of the different gear changes was unified. Although, when the agent was trained, the gear change was accepted as successful if the position error was below $0.1mm$. Therefore, during comparison, the shift time has been defined as the time interval between the emergence of a new request and reaching $-0.1mm$.

The LQR controller is not able to achieve Neutral to Low gear changes according to the given qualitative requirements. As the system is highly nonlinear, the LQR controller switches between different linear State-Space models of the system, but due to the extremely high speed of the system, this switching may cause transients around Neutral, which cannot be handled with the given control frequency.

Table 6 shows the performance of both the PID controller and the PG agent for the analyzed gear change. In case of the PID controller, the shift time is inversely proportional to the supply pressure, and the controller fails the requirements at $7bar$ supply pressure, while the LTI controllers have not been tested over $9bar$, hence the comparison is limited to $7bar$ - $9bar$ supply pressure. On higher supply pressures, the PID algorithm is clearly better as it achieves faster gear changes with no overshoot. However, as the supply pressure decreases, its performance suddenly drops, and the advantage of the PG algorithm becomes clear. The PID controller was tuned to maximize its performance on a dedicated supply pressure, namely $9\ bar$, while the PG algorithm learned to maximize its reward on a much wider supply pressure range. Hence, around $9\ bar$, the PID controller can perform better, but the higher the deviation from this nominal value, the worse its performance gets. On the other hand, the PG algorithm changes its strategy depending on the supply pressure, thus it has a much more balanced performance. It is also worth mentioning, that under $6bar$ supply pressure, the PID controller cannot reach the request as the piston will be stuck between Neutral and Low, while the PG agent meets the requirements even at $4bar$ supply pressure. Within the analyzed interval the performance of the PG algorithm is almost constant, while in case of the PID controller, the shift time increased by approximately 150%.

TABLE 6. Comparison of the PID and PG control performances.

Supply pressure [bar]	Shift time [ms]		Overshoot [mm]		No. of valve cmd [-]	
	PID	PG	PID	PG	PID	PG
7	132	74	0	0.03	1	3
7.5	88	75	0	0.02	1	3
8	72	77	0	0	2	2
8.5	53	75	0	0	2	2
9	52	74	0	0	2	2

The comparison of the two controllers shows that an LTI controller can be designed to have better performance under nominal operational conditions, but learning-based controllers can learn to operate under a much wider range.

VI. CONCLUSION

This paper presents the machine learning control of an electro-pneumatic actuator in an automated manual transmission. Technically, this task is a set-valued position control of a nonlinear delayed system. The presented research has three main contributions:

First, a new approach for neural network aided guiding of MCTS search, what is called PG-MCTS, is presented. The known methods, such as the Planning agent used in this paper, apply values and policy networks to supplement the UCT function, and replace the rollout phase of the MCTS. In contrast, in the presented PG-MCTS, only a policy network serves this purpose. As shown, this method enhances the efficiency of the agent with short horizon prediction and remains scalable for resources.

The second contribution is the comparison of various methods. Two pure RL agents (DDQN and PG), an MCTS for benchmark and training data generation for supervised learning, the SL agent itself, the classic Planning agent, and the presented PG-MCTS are compared based on their performance.

Last, the network of a selected agent is implemented on a real gearbox, and shown, that its performance is better than the classic LTI controllers designed with LQR or PID approaches. Naturally, the cause of this difference is that classical control has limited possibilities for dealing with nonlinearities of the system and the set-valued actuator inputs.

The research has shown that a purely heuristic MCTS algorithm can effectively solve the control task, though it requires a large number of rollouts, making real-time applicability impossible. The supervised learning based neural network agent trained on the data provided by MCTS produces approximately the same performance, however, in a real-time manner, contrary to RL based methods, its training is more complicated because of the generation and preparation of the training data. Moreover, it results in inferior performance compared to RL techniques.

The Planning agent and the PG-MCTS are superior to the others presented, and they show reinforcement learning can push the reachable performance-level above and beyond. However, the restrictions represented by the evaluation of the algorithms on an automotive control unit highlighted that not the performance and real-time capabilities are the only factors that matter. Consequently, during the deployment of such methods, the memory and computational resource needs have to be considered, because the cost of such controllers is crucial in the scale of mass production. The Planning agent's problematic and computationally exhaustive training seems overwhelming, thanks to the drawling planning before every move and the trial-and-error-based operation of the whole RL field.

The scalability of the PG-MCTS algorithm enables to handle the trade-off between agent performance and resource needs during the prediction phase by adjusting the planning length to the time-frame provided, even though on the actual plant, the PG-MCTS could not fit in the provided ECU. In contrast, the shortcomings in performance are entirely balanced by efficiency in the case of the PG agent, making it the most feasible trade-off for the actual application.

The presented PG-MCTS method with the modified UCT algorithm exploits the unique features of the Policy-based agents. Consequently, the design patterns and mostly the comparison of their behavior are suitable for similar control problems.

REFERENCES

[1] C. Fritschy and S. Spinler, "The impact of autonomous trucks on business models in the automotive and logistics industry—A Delphi-based scenario study," *Technol. Forecasting Social Change*, vol. 148, Nov. 2019, Art. no. 119736.

[2] J. K. Tar, K. Lorincz, and R. Kovacs, "Adaptive control of an automatic convoy of vehicles," in *Proc. 11th Int. Conf. Intell. Eng. Syst.*, Jun. 2007, pp. 21–26.

[3] C. Lienke, C. Wissing, M. Keller, T. Nattermann, and T. Bertram, "Predictive driving: Fusing prediction and planning for automated highway driving," *IEEE Trans. Intell. Vehicles*, vol. 4, no. 3, pp. 456–467, Sep. 2019.

[4] J. K. Kolb, G. Nitzsche, and S. Wagner, "A simple yet efficient path tracking controller for autonomous trucks," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 307–312, 2019.

[5] Á. Török, Z. Szalay, G. Uti, and B. Verebélyi, "Rerepresenting autonomated vehicles in a macroscopic transportation model," *Periodica Polytechnica Transp. Eng.*, vol. 48, no. 3, pp. 269–275, 2020.

[6] J. Li, X. Mei, D. Prokhorov, and D. Tao, "Deep neural network for structural prediction and lane detection in traffic scene," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 690–703, Mar. 2017.

[7] Y. Y. Ye, X. L. Hao, and H. J. Chen, "Lane detection method based on lane structural analysis and CNNs," *IET Intell. Transp. Syst.*, vol. 12, no. 6, pp. 513–520, Aug. 2018.

[8] W. Song, Y. Yang, M. Fu, Y. Li, and M. Wang, "Lane detection and classification for forward collision warning system based on stereo vision," *IEEE Sensors J.*, vol. 18, no. 12, pp. 5151–5163, Jun. 2018.

[9] J. George, L. Mary, and R. K S, "Vehicle detection and classification from acoustic signal using ANN and KNN," in *Proc. Int. Conf. Control Commun. Comput. (ICCC)*, Dec. 2013, pp. 436–439.

[10] A. Feher, S. Aradi, and T. Becsi, "Q-learning based reinforcement learning approach for lane keeping," in *Proc. IEEE 18th Int. Symp. Comput. Intell. Informat. (CINTI)*, Nov. 2018, pp. 31–36.

[11] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *Proc. Amer. Control Conf. (ACC)*, May 2017, pp. 4914–4919.

[12] B. Németh, "Robust LPV design with neural network for the steering control of autonomous vehicles," in *Proc. 18th Eur. Control Conf. (ECC)*, Jun. 2019, pp. 4134–4139.

[13] D. V. Ngo, T. Hofman, M. Steinbuch, A. Serrarens, and L. Merkx, "Improvement of fuel economy in power-shift automated manual transmission through shift strategy optimization—An experimental study," in *Proc. IEEE Vehicle Power Propuls. Conf.*, Sep. 2010, pp. 1–5.

[14] X. Zhu, H. Zhang, J. Xi, J. Wang, and Z. Fang, "Robust speed synchronization control for clutchless automated manual transmission systems in electric vehicles," *Proc. Inst. Mech. Eng. D, J. Automobile Eng.*, vol. 229, no. 4, pp. 424–436, Mar. 2015.

[15] M. Van Damme, P. Beyl, B. Vanderborght, R. Van Ham, I. Vanderniepen, R. Versluys, F. Daerden, and D. Lefeber, "Modeling hysteresis in pleated pneumatic artificial muscles," in *Proc. IEEE Conf. Robot., Autom. Mechatronics*, Sep. 2008, pp. 471–476.

[16] H. Aschemann and D. Schindele, "Comparison of model-based approaches to the compensation of hysteresis in the force characteristic of pneumatic muscles," *IEEE Trans. Ind. Electron.*, vol. 61, no. 7, pp. 3620–3629, Jul. 2014.

[17] T. Vo-Minh, T. Tjahjowidodo, H. Ramon, and H. Van Brussel, "A new approach to modeling hysteresis in a pneumatic artificial muscle using the Maxwell-slip model," *IEEE/ASME Trans. Mechatronics*, vol. 16, no. 1, pp. 177–186, Feb. 2011.

[18] A. Pujana-Arrese, A. Mendizabal, J. Arenas, R. Prestamero, and J. Landaluze, "Modelling in modelica and position control of a 1-DoF set-up powered by pneumatic muscles," *Mechatronics*, vol. 20, no. 5, pp. 535–552, Aug. 2010.

[19] B. Szimandl and H. Németh, "Closed loop control of electro-pneumatic gearbox actuator," in *Proc. Eur. Control Conf. (ECC)*, Aug. 2009, pp. 2554–2559.

[20] K. Balasubramanian and K. S. Rattan, "Fuzzy logic control of a pneumatic muscle system using a linearing control scheme," in *Proc. 22nd Int. Conf. North Amer. Fuzzy Inf. Process. Soc. (NAFIPS)*, 2003, pp. 432–436.

[21] S. W. Chan, J. H. Lilly, D. W. Repperger, and J. E. Berlin, "Fuzzy PD+I learning control for a pneumatic muscle," in *Proc. 12th IEEE Int. Conf. Fuzzy Syst. (FUZZ)*, May 2003, pp. 278–283.

[22] D. Zhang, X. Zhao, and J. Han, "Active model-based control for pneumatic artificial muscle," *IEEE Trans. Ind. Electron.*, vol. 64, no. 2, pp. 1686–1695, Feb. 2017.

[23] G. Bone, M. Xue, and J. Flett, "Position control of hybrid pneumatic–electric actuators using discrete-valued model-predictive control," *Mechatronics*, vol. 25, pp. 1–10, Nov. 2014.

[24] H. Qi, G. M. Bone, and Y. Zhang, "Position control of pneumatic actuators using three-mode discrete-valued model predictive control," *Actuators*, vol. 8, no. 3, p. 56, Jul. 2019.

[25] A. Grancharova and T. A. Johansen, "Explicit model predictive control of an electropneumatic clutch actuator using on/off valves and pulse-width modulation," in *Proc. Eur. Control Conf. (ECC)*, Aug. 2009, pp. 4278–4283.

[26] E. J. Barth, J. Zhang, and M. Goldfarb, "Sliding mode approach to PWM-controlled pneumatic systems," in *Proc. Amer. Control Conf.*, vol. 3, May 2002, pp. 2362–2367.

[27] H. Sande, T. A. Johansen, G.-O. Kaasa, S. R. Snare, and C. Bratli, "Switched backstepping control of an electropneumatic clutch actuator using on/off valves," in *Proc. Amer. Control Conf.*, Jul. 2007, pp. 76–81.

[28] G. M. Bone and X. Chen, "Position control of hybrid pneumatic-electric actuators," in *Proc. Amer. Control Conf. (ACC)*, Jun. 2012, pp. 1793–1799.

[29] B. Szimandl and H. Németh, "Sliding mode position control of an electro-pneumatic clutch system," *IFAC Proc. Volumes*, vol. 46, no. 2, pp. 707–712, 2013.

[30] L. Ma, G. Zong, X. Zhao, and X. Huo, "Observed-based adaptive finite-time tracking control for a class of nonstrict-feedback nonlinear systems with input saturation," *J. Franklin Inst.*, Sep. 2019.

[31] T. Bécsi, S. Aradi, Á. Szabó, and P. Gáspár, "Policy gradient based reinforcement learning control design of an electro-pneumatic gearbox actuator," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 405–411, 2018.

[32] B. Kovari, T. Becsi, A. Szabo, and S. Aradi, "Policy gradient based control of a pneumatic actuator enhanced with Monte Carlo tree search," in *Proc. 6th Int. Conf. Mechatronics Robot. Eng. (ICMRE)*, Feb. 2020, pp. 177–182.

[33] A. Szabo, T. Becsi, P. Gaspar, and S. Aradi, "Control oriented modeling of an electro-pneumatic gearbox actuator," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2018, pp. 2623–2628.

[34] A. Szabo, T. Becsi, and S. Aradi, "Measurement based validation of an electro-pneumatic gearbox actuator," *Periodica Polytechnica Transp. Eng.*, pp. 1–7, 2019. [Online]. Available: https://pp.bme.hu/tr/article/view/14265, doi: 10.3311/PPtr.14265.

[35] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artif. Intell.*, vol. 175, no. 11, pp. 1856–1875, Jul. 2011.

[36] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer go: Monte Carlo tree search and extensions," *Commun. ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012.
[37] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Eur. Conf. Mach. Learn.* Berlin, Germany: Springer, 2006, pp. 282–293, doi: 10.1007/11871842_29.
[38] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*. [Online]. Available: http://arxiv.org/abs/1604.07316
[39] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3207–3214.
[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602
[41] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*. [Online]. Available: http://arxiv.org/abs/1511.06581
[42] A. Barto and R. S. Sutton, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.
[43] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[44] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.
[45] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971
[46] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.
[47] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.
[48] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.
[49] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[50] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *J. Roy. Statist. Soc., B Stat. Methodol.*, vol. 61, no. 3, pp. 611–622, 1999.
[51] Vector. *VN1600 Interface Family Manual*. Accessed: Apr. 16, 2020. [Online]. Available: https://assets.vector.com/cms/content/products/VN16xx/docs/VN1600_Interface_Family_Manual_EN.pdf
[52] Vector. *CANape Product Information*. Accessed: Apr. 16, 2020. [Online]. Available: https://assets.vector.com/cms/content/products/canape/Docs/CANape_ProductInformation_EN.pdf
[53] A. Szabo, T. Becsi, and P. Gaspar, "Control design and validation for floating piston electro-pneumatic gearbox actuator," *Appl. Sci.*, vol. 10, no. 10, p. 3514, May 2020.

TAMÁS BÉCSI (Member, IEEE) received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics, Budapest, Hungary, in 2002 and 2008, respectively.

Since 2005, he has been an Assistant Lecturer and since 2014, he has also been an Associate Professor with the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics. His research interests include linear systems, embedded systems, traffic modeling, and simulation. His research and industrial works have involved railway information systems and vehicle control.

ÁDÁM SZABÓ received the B.Sc. and M.Sc. degrees from the Budapest University of Technology and Economics, Budapest, Hungary, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the Department of Control for Transportation and Vehicle Systems.

His research and industrial works have involved modeling and control of heavy-duty transmission systems.

BÁLINT KŐVÁRI received the B.Sc. degree in vehicle engineering from the Budapest University of Technology and Economics, Budapest, Hungary, in 2018, where he is currently pursuing the M.Sc. degree with the Faculty of Autonomous Vehicle Control Engineering.

His research interests include artificial intelligence, machine learning, vehicle dynamics, and mechatronics.

SZILÁRD ARADI (Member, IEEE) received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics, Budapest, Hungary, in 2005 and 2015, respectively,

Since 2016, he has been a Senior Lecturer with the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics. He is currently working with the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics. His research interests include embedded systems, communication networks, vehicle mechatronics, and reinforcement learning. His research and industrial works have involved railway information systems, vehicle on-board networks, and vehicle control.

PÉTER GÁSPÁR received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics (BME), Faculty of Transportation Engineering and Vehicle Engineering (KJK), in 1985 and 1997, respectively, and the D.Sc. degree in control from the Hungarian Academy of Sciences (MTA), in 2007.

Since 1990, he has been a Senior Research Fellow with the Institute for Computer Science and Control (SZTAKI). Since 2016, he has also been a Research Professor. In 2004, he became the Head of the Vehicle Dynamics and Control Research Group and then in 2017, he became the Head of the Systems and Control Laboratory, SZTAKI. He was habilitated at the BME, in 2008, and he was appointed as the University Professor. Since 2013, he has also been the Head with the Department of Control for Transportation and Vehicle Systems (KJIT), BME KJK. His research interests include linear and nonlinear systems, robust control, multi-objective control, system identification, and identification for control and artificial methods. His research and industrial works have involved mechanical systems, vehicle structures, and vehicle dynamics and control. Since 2016, he has also been a Corresponding member of MTA. He is also a member of the IFAC Automotive Control and Transportation Systems Technical Committee, and the Chair of the International Federation of Automatic Control (IFAC) Hungary National Member Organization.