Controller design revisited NEATly: Controlling an electric motor in a multi-objective setting under varying load using NeuroEvolution of augmenting Topologies

Nick Dienemann, *Hochschule Karlsruhe*

***Abstract*—PID controllers and their extensions dominate the control systems of today, due to their simple design and mathematical motivation. The success of PID controllers is based on the linearization of physical systems and a one-dimensional objective, namely controlling for maximum precision. However, highly dynamical reference functions render a linearization of the physical system difficult and modern control tasks often seek to not only control precisely but demand secondary control objectives like efficiency, safety or component sustainability. These boundary conditions construct a high dimensional optimization space for which finding an optimal solution mathematically becomes infeasible. This work examines neuro-evolutionary algorithms as a potential alternative to solve these high dimensional, often non-linear optimization problems. The approach is applied to the control task of an electric motor and compared to manually designed PID controllers and PID controllers whose parameters are tuned with an evolutionary algorithm.**

***Index Terms*—Control System Engineering, NEAT, multi-objective optimization, Permanently Excited DC motor**

I. Introduction

PID controllers are the most widely used controller types for control systems [8]. Due to their well studied behavior and simple design they can be deployed quickly to a wide variety of different control problems. For linear systems, an optimal PID controller can be derived that theoretically reaches arbitrary precision. The applicability of this optimal solution mainly depends on two factors: linearity of the system in question and precision as the only optimization goal. While most systems in the real world are not actually linear, the classical control theory has used the technique of linearization to locally approximate a non-linear system with a linear counterpart. This approximation is based on expanding non-linear terms into a Taylor-Series around an operating point and then only considering the linear terms. As is well-known for the full Taylor expansion, it is an exact solution exactly in the operating point and becomes increasingly inaccurate as the distance from that operating point increases. This inaccuracy is amplified by not considering any terms of order two and higher. Consequently, this linearization may work for systems that are supposed to be controlled within a very small interval but fails entirely for target functions with high dynamics. The latter constraint, namely that precision is the only optimization goal, is becoming less and less realistic. A rising demand for effective, environment friendly and long lasting production is driving the development of control systems that do not only control precisely and robustly, but also do meet numerous boundary conditions. Combining the inaplicability of linearization and increasing complexity of the optimization goal it is self evident that new methods need to be developed. As the task boils down to a non-linear optimization problem in a high dimensional space, neural networks are a promising contender to fullfill the challenging requirements in modern control systems.

The training of the neural network is not as trivial as it is for classical supervised settings. As the optimal control behavior is unknown a priori, no labeled data can be derived rendering classical supervised training schemes inapplicable. Fortunately, Reinforcement Learning (RL) is designed specifically for problems where no supervised data is available but data can be obtained by interacting with an environment. RL uses so called observations obtained from interacting with an environment, e.g. an electric motor, and a manually designed objective function to train an agent, e.g. a controller. It is this interaction based learning that enables the application of a neuro-evolutionary algorithm. A Neuro-evolutionary algorithm is a directed search that tunes the parameters of a neural network by producing a quantity of potential solution candidates randomly and then refining them using operators similar to those seen in evolutionary processes in the real world. A more detailed description of the algorithm is given in III-A.

II. Related Work

According to [2] the term ”classical control theory” dates back to the 1960s refering to ”the body of knowledge developed during the years 1930 to 1955”([2], prefix p.8). Within the last century, classical control theory has developed massively and overcome numerous hurdles. Self evidently it has also developed approaches to address the problems named in the introduction, namely non-linear systems and multidimensional optimization goals. A popular approach of tackling the problem of non-linear systems is to use fuzzy control in combination with PID controllers [20] [9] [17].

Nick Dienemann was with the Department of Electrical Engineering and Information Technology, Hochschule Karlsruhe - University of Applied Sciences, Karlsruhe.

The source code is available at:
https://github.com/NickDienemann/projektarbeit

Manuscript received April 19, 2005; revised August 26, 2015.

However, modern approaches also yield promising results by completely leaving the PID controller behind. [4] use fuzzy logic in combination with a state feedback controller. [19] extend the robust integral of the sign of the error feedback control to the working condition of output feedback. However, all of these approaches and the resulting control strategies are not easily understandable and may thus lack behind less performant but simpler approaches independent from their magnificant performance. When it comes to studying the control task of an electric motor with PID controllers research is conducted towards robustness of the controllers as in [6]. [16] on the other hand investigate the reduction of torque ripples when controlling a permanent magnet synchronous motor (PMSM) by extending the PI structure with a resonant controller-based model predicitve control (MPC).

Eventhough, [4] use neural networks to model uncertainties in the system, none of the above work directly uses neural networks as their controller. When it comes to using neural networks for the controller specifically within the electric motor domain the group of [14] are leading experts. Their research focuses on using Deep Reinforcement Learning, Actor-Critic based methods to be specific, for the control task. While their first work [14] was dedicated to a proof of concept of RL based controller design for electric motors, the authors put more emphasis on the challenges and according adaptations when deploying the RL-controller in the real world in [3], a work that they improved on further in [13]. This line of research has yielded promising results, both in the simulation and more importantly on real world systems. However, it is focused merely around the controller design based on the actor-critic methods and its adaptations.

Another line of research is the use of evolutionary algorithms (EA) for controller design [11] [7] [12]. Using evolutionary algorithms for the controller design has the advantage that an optimization for multiple objectives can be solved instead of the single optimization goal in classical control theory. The works of [11] [7] [12] focus on tuning the parameters of a PID (or PI) controller during this evolutionary process. This limits the produced solutions to the linear behavior inherent to the PID controller thus rendering their applicability to nonlinear control problems questionable. A slightly different approach is shown in [10] who loosen this constraint by optimizing the membership functions of a fuzzy controller using an evolutionary algorithm.

An alternative approach to loosen the constraint of linearity imposed by the underlying PID structure is presented in this work. This is achieved by using a combination of neural networks and evolutionary algorithms, i.e. neuro-evolutionary algorithms. While [18] also apply neuro-evolution to control a four-legged robot using Evolution of Network Symmetry and mOdularity (ENSO), their modular approach constraints the network structure to the fixed network architectures. Similarly, [5] present a novel neuroevolution method called CoSyNE that optimizes the weights of a user-predefined network for the control of non-linear problems. Given that both of the available approaches predefine the network architecture, the covered solution space is limited considerably making it less attractive especially for nonlinear problems. Instead, in this work NeuroEvolution of Augmenting Topologies (NEAT) [15] is used to develop the neural network representing the controller. NEAT develops both the networks weights as well as the networks architecture thus harnessing the full extend of the solution space. To the best of the authors knowledge, this work is the first to apply NEAT to the problem of controller design. For the control task the control of an electric motor is chosen as it contains both linear and nonlinear variants. In summary, the contributions of this work are the following:

1) Examine the applicability of NEAT to control tasks on the example of an electric motor.
2) Evaluate NEAT for multi-objective optimization in controller design.
3) Provide an extension to the Gym Electric Motor (GEM) [1] framework to allow versatile training and validation of classical and modern controllers.

III. Methodology

A. NeuroEvolution of augmenting Topologies

NEAT was originally presented in the work of Stanley et al [15]. Like all evolution based algorithms, NEAT is a directed search as its overall goal is to find solution candidates that yield the highest objective score according to a user defined objective function, the so called fitness function. Ideally, this function represents an objective function describing how well a solution candidate solves a specified task or in other words how well it is adapted to (or fit to) a given environment. As with most evolutionary algorithms, an initial set of solution candidates is created randomly at the beginning. Based on this initial set of solution candidates, called population, the algorithm iteratively refines the solutions using operations similar to those found in real-world evolutionary processes, namely mutation and crossover. To apply these operations, the solution candidates, in this case neural networks, must first be encoded into a genotype. A genotype refers to the set of all genes, each describing a single aspect of the underlying object, e.g. a single gene might represent a single node of a neural network. This abstract description is then suitable to undergo mutation and crossover operations, allowing for further refinement of the solution candidate it describes. Each genotype has a corresponding phenotype that is the actual object of interest, in this case the neural network itself. The execution of NEAT runs in the usual manner for evolutionary algoritms. For each iteration, called generation, a fitness value is assigned to each individual of the current population. Then, based on these fitness values an individual is either kept as it is, changed or removed entirely. Individuals with a high fitness are more likely to be kept, a concept called elitism to ensure exploitation, while the worst performing individuals are more likely to be removed. Those solution candidates that are chosen to be changed are either mutated, a process where some genes of the corresponding genotype are changed randomly or crossed over with another individual, thus yielding a new individual that possesses genes from both its parents. This step yields a new population which will then undergo the same two-step process. The algorithm either stops after a given number of generations or when an individual meets a predefined fitness threshold.

For a complete in-depth description, that also explains important additions to the concept described above like speciation, the reader is refered to [15].

B. Gym Electric Motor

For the simulations of the electric motors in this work the framework Gym Electric Motor (GEM) [1] is used. This framework provides multiple environments based on Open AI's Gym package designed specifically for the design of controllers with RL. Not only does it contain a wide variety of different electric motor types but also comes with an entirely configurable system, i.e. voltage supply, power electronic converter, electric motor and mechanical load. Additionally, different reference functions are available and the controllers behavior can be visualized. This work extends the available functionality by a more dedicated approach to reward functions, deeper analysis of the controllers behavior with the help of step-based analyses and most importantly a training, validation and guidance scheme explained in detail in section III-C. Furthermore, the extension also unifies the configuration of the system using simple configuration files instead of in-code changes to allow convenient configuration of agent, training, validation, guidance and environment. Finally, it allows to run and evaluate multiple experiments in parallel on a computation system level.

C. The framework: Training, validation and guidance scheme

The extended framework developed in this work aims to supply functionality to train and validate controllers in a RL fashion. It defines a training and validation scheme that is suitable for all kinds of algorithms and while it is only used for NEAT, an EA and a classical controller it is equivalently suited for other algorithms, e.g. Q-Learning, Actor-Critic etc. . To adress this generalizability figure 1 describes the input from the algorithm as an entity, e.g. a population in the case of NEAT, a single agent in the case of Actor-Critic, etc. . In the following, the main execution scheme of the framework is presented according to its visual representation in figure 1. Within the framework, there are three types of iterations: Training, Validation and Guidance. Following the usual approach in RL, each of those iterations is centered around a so called play session. Within this play session one agent (or multiple depending on the algorithm of choice) is used as a controller within a set of environments for a specified number of steps. This play session yields episodes for each environment and those episodes are then processed further according to the type of iteration. As the name suggests, the training iteration is dedicated to the training of the agent (or agents). In the case of NEAT this would mean that the episodes yielded for each genome in the current population would be used to assign a fitness value to each genome. The population that now contains genomes with fitnesses assigned to them would then undergo a single step of evolution resulting in the population of the next generation as the output of the training iteration. Generally speaking, the output of the training iteration is always a trained version of the input. After a predefined number of training iterations the trained entity is passed on to either a validation iteration or/and a guidance iteration. The first step of both of these iterations is to extract the best agent within this entity, for NEAT this would be the genome with the highest fitness and for Actor-Critic it would simply be the agent itself. The goal of the validation iteration is to evaluate the best agent's performance on a different set of environments. As the environments are not used during training, this iteration offers great functionality to access the agents generalizability. Furthermore, due to its configurable execution frequency, that is usually chosen to be number of magnitudes lower than for the training iterations, this iteration allows for an extended use of the agent without considerably effecting overall execution time. To gain these insights, the validation iteration uses the generated episodes for the creation of visualizations and analyses. Recall that the agent used within this iteration remains completely unchanged and the sole purpose is to give visual and numerical insights into the behavior of the current best agent. And lastly, there is the so called guidance iteration, which is a novel technique for environment exploration and is presented in detail in section III-F. Basically it allows to create new training environments for the next traning iteration that are designed specifically to guide the training towards a certain direction. This direction is derived by the episodes yielded by the best agent when deployed onto the guidance environments and usually corresponds to creating environments that the current best agent performs poorly on.

The scheme described above does not only allow to train the given entity but also to get deep insights into its behavior over the course of training.

D. Reward function

The reward function is used to obtain an objective score that rewards an agent for a desired behavior and potentially punishes it when behaving in an undesired way. A behavior can be described as one point in an abstract behavior space and it is the task of the reward function to map this point in behavior space onto a real number, indicating the adequacy of that behavior with respect to some high level objective, e.g. precision, efficiency etc. . However, to be able to formulate this in a machine understandable way, the point in behavior space needs to be quantified. It is customary in RL to approximate the behavior by a set of current state s, action a taken by the agent and next state s'. On this high level, the reward function can be defined as follows.

$$\begin{aligned} G(b) &: \mathbb{B} \rightarrow \mathbb{R}^{(k)} \\ r(s,a,s') &: \mathbb{R}^{(k)} \rightarrow \mathbb{R}^{(1)} \end{aligned} \tag{1}$$

Where $\mathbb{B}$ denotes the abstract behavior space, b one point in this behavior space, $G(b)$ an unknown function that maps b onto a k-dimensional set $\{s,a,s'\}$ and r the reward function. For the remainder of this work, the most general form of a reward function is defined as a composite function of n reward terms each tackling a single aspect of a higher level objective:

$$r(s,a,s') = \lambda_1 \cdot r_1(s,a,s') + ... + \lambda_n \cdot r_n(s,a,s') \tag{2}$$

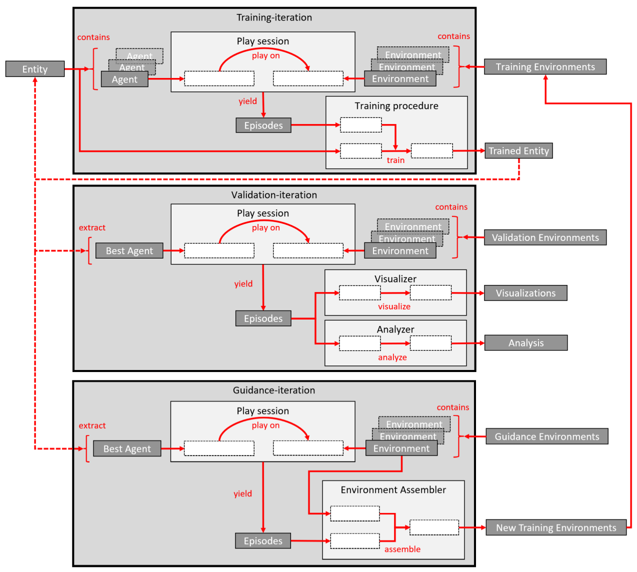


Fig. 1: Training, validation and guidance scheme. The user supplies an entity and the environments for each type of iteration. The given entity then undergoes the shown three-step process repeatedly not only to be trained but also to allow the user to access performance indicators during training and get full visualisations of the entities behavior.

Of course, any lower dimensional form of this equation that does not incorporate a, s' or both also defines a reward function.

Logically, the goal of the optimization is to maximize the total reward aquired and thus the reward function is the main instrument to guide the optimization process. We formulate the following attributes that are integral for a healthy reward function that guides towards optimal behavior. Firstly, as a matter of defintion, reward functions output a higher score for better behavior and a lower score for a behavior that is worse. Secondly, in case the reward function is composed of multiple components, each of those components should be normalized, so that their importance to the higher level objective can be configured comprehensively by adapting the corresponding weights of the individual terms. Thirdly, in an ideal case the reward as a function of the behavior should be starting from 0 at the point of worst possible behavior and linearly increase to 1 at the point of optimal behavior. In other words, the difference in reward between two sets of $\{s, a, s'\}$ should be proportional to the improvement in the underlying behavior space. Note that this is not necessarily the same as a proportionality of the distance in the $\mathbb{R}^{(k)}$ space. As an example, the L1-distance between a reference and the current value may be chosen to quantify the precision of a controller. If this distance would be used directly as a reward, then the reward would be proportional to the distance itself. However, the L1-distance itself is bound to the optimal value of 0, thus saturating the reward term as it gets closer and closer towards the optimal value. In a single-objective setting this would not really be a problem, as the absolute scale of the reward does not matter when optimizing for this single score. For a multi-objective setting however, this reward saturation is very problematic, as the saturated reward term will be dominated by other non-saturated reward terms leading to an optimization that continuously favors maximising the non-saturated reward. This discrepency has to be taken into account when defining the corresponding reward term.

To sum it up, we claim that a healthy reward function should be:

1) Monotonically increasing with respect to an improvement in behavior space. Let $\{s_1,a,s_1'\}$ and $\{s_2,a_2,s_2'\}$ be two state-action-next state pairs, where the latter describes a behavior that is better than the former regarding a high level objective. Then:
$$r(s_2, a_2, s_2') > r(s_1, a_1, s_1') \tag{3}$$
2) Composed of normalized individual reward terms. Let $\{r_1, r_2, ..., r_n\}$ be a set of reward terms of a composite reward function r, then:
$$0 \leq r_i(s, a, s') \leq 1, \forall i \in \{1, 2, ..., n\} \tag{4}$$
3) Proportional to the behavior space. Let $\{s_1,a,s_1'\}$ and $\{s_2,a_2,s_2'\}$ be two different state-action-next state pairs, then:
$$\begin{aligned} r(s_2, a_2, s_2') - r(s_1, a_1, s_1') \propto & G^{-1}(s_2, a_2, s_2') \\ & - G^{-1}(s_1, a_1, s_1') \\ & = b_2 - b_1 \end{aligned} \tag{5}$$

In order to simplify the design of the reward function, the extended framework of this work offers functionality to convert distance metrics into reward functions(1), normalize them(2) and address the proporionality to the behavior space(3). This is achieved by using a reward-finalization function that comes with options for all three cases.

In case the function of quantification chosen by the designer of the reward term behaves like a distance d, thus violating attribute (1), a naive approach would be the inversion:
$$r = -1 \cdot d \tag{6}$$
This can then be normalized with respect to the minimum and maximum value of the distance to also accomplish attribute (2). For a non-saturating distance or a single objective setting, this would be sufficient. However, in case the distance is saturating when approaching optimal behavior we propose the following distance-to-reward conversion function, called inverse Logarithmified Linear Unit or iLogLU for short:
$$iLogLU(x) = \frac{1}{ln(s)} \cdot \begin{cases} \frac{x-k}{k-l} & x \geq k \\ ln(\frac{x-l}{k-l}) & else \end{cases} \tag{7}$$
Where s,k and l are hyperparameters to choose. Parameter $s \in (0, 1)$ determines the steepness of the function and k describes the x value at which the transition from linear to logarithmic function takes place. l determines the value at which the iLogLu approaches infinity. Let x be a distance metric taking values in $[a, b]$ that saturates at value a, meaning that it approaches the value in $a + e^{-x}$ fashion but behaves linearly on an interval $[c, b]$ with $c > a$. Then the parameters of iLogLU should be chosen to $k = c$ and $l = a - \zeta$, where ζ

is a small positive constant to ensure numerical stability (set to 0.01 for the experiments). This can then be normalized for the interval $[a, b]$. In case the distance metric already fullfills attribute (1), then a non-inverse version, LogLU, of the function above can be used instead:

$$LogLU(x) = \frac{1}{ln(s)} \cdot \begin{cases} \frac{k-x}{l-k} & x \leq k \\ ln(\frac{l-x}{l-k}) & else \end{cases} \tag{8}$$

A sample plot of the NiLogLU can be seen in figure 2.

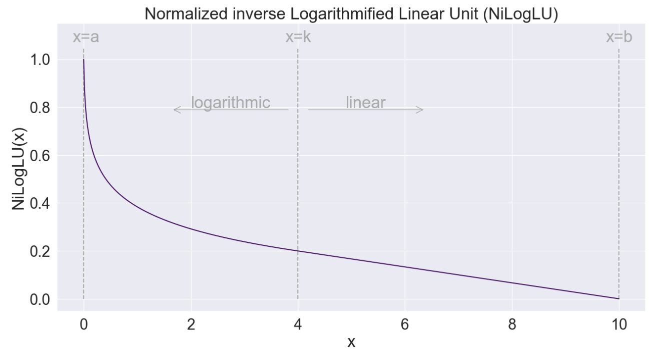


Fig. 2: Normalized inverse Logarithmified Linear Unit (NiLogLU) for parameters $k = 4$, $a = 0$, $b = 10$, $l = a - \zeta = 0 - 0.01$ and $s = 0.1$. Note the high gradient when approaching the lower bound that compensates for a saturating behavior of the distance meassure concerning behavior space.

E. Fitness function

While the reward function is necessary for all kinds of RL, the fitness function is proprietary to evolutionary algorithms, i.e. both the standard EA and also NEAT. As stated before, the reward function aims to give an objective score indicating the adequacy of the agents behavior w.R.t. some higher level objective for each step played on an environment. Similarly, the fitness function assigns an objective score to a genome to describe how well its corresponding phenotype is adapted to solving a specified task. So while the reward function assigns a reward to a state-action-next state transition indicating behavior adequacy, the fitness function assigns a fitness to a genome indicating the genomes phenotypes level of adaptation to the task at hand. However, those two scores are highly correlated, as an agent that yields transitions with an adequate behavior, i.e. high rewards, can also be considered as very well adapted to solving the overall task at hand, i.e. high fitness. In fact, for both the EA as well as NEAT the fitness is calculated in the following manner for the general case. Let K = $\{K_1, K_2, ..., K_l\}$ be a set of episode sets, where $K_i = \{E_{i1}, E_{i2}, ..., E_{in}\}$ denotes a set of n different Episodes E played on the i-th of l training environments. Each episode contains a variable number of at max m steps st, such that $E_{in} = \{st_{in1}, str_{in2}, ..., st_{inm}\}$. Further, let $st[r]$ denote the reward of a single state-action-next state transition. Then the fitness F of the genome is calculated by:

$$F = \frac{1}{l} \cdot \sum_{i=1}^{l} \frac{1}{m} \sum_{j=1}^{m} \frac{1}{n} \sum_{k=1}^{n} st_{ijk}[r] \tag{9}$$

F. Guidance

Generally speaking, there are two types of exploration: agent parameter space exploration and environment exploration. While the former describes the exploration of the space that is spanned by the tunable parameters of the agent in question, the latter deals with exploring the environment that the agent interacts with. Guidance seeks to improve the capability of NEAT, and population based optimization algorithms in general, to explore the entire environment. This is achieved by using guidance iterations that are based on long play sessions with only the best agent known at the time on a set of guidance environments. As only a single agent performs the guidance iteration it is feasible to perform very long play sessions that allow for an extensive exploration of the environment without introducing a high computation time increase. The episodes yielded during this extended play session are then interpreted by the so called environment assembler. The environment assembler extracts an environment state that the best agent performs the worst on and creates a new environment that is setup slightly before that state was reached. The environment thus constructed is then added to the training environments for all upcoming training iterations. Recall that those training environments are used for the training of the entire population. Thereby, the information of the environment exploration is propagated indirectly to the entire population. This allows to use as small of a number of training steps as possible without the penalty of to little environment exploration, as events that only occur after a larger number of steps are reinduced into the training process after each guidance iteration.

IV. Experiments

The experiments conducted aim on comparing three approaches of controller design, namely a a manually designed cascaded PID controller (from now on refered to as classical controller), a cascaded PID controller whose parameters are tuned with an evolutionary algorithm (hereafter named ea-controller) and a controller created with NEAT (called neat-controller for the remainder of this work). Within those experiments, the classical controller and the ea-controller serve as a benchmark for the neat-controller in terms of precision and energy efficiency.

A. Environment

As a base environment a permanently excited DC motor is chosen because when no mechanical load is applied the environments ODE system is completely linear, as can be seen in equation (10) based on the work of [1]. This linearity allows the manual design of the classical controller even for a highly dynamic reference function.

$$\begin{pmatrix} \frac{di}{dt} \\ \frac{d\omega}{dt} \end{pmatrix} = \begin{pmatrix} \frac{1}{L_A}(-\Psi'_E \omega - R_A i + u) \\ \frac{1}{J}(\Psi'_E i - T_L(\omega)) \end{pmatrix} \tag{10}$$

A full description of the environment used can be found in appendix D.

B. Setting

This experiment is divided into the following four parts that vary in the mechanical load applied and the reward function used:

1) no load + no reward for input energy
2) no load + reward for input energy
3) with load + no reward for input energy
4) with load + reward for input energy

For each of these settings, the classical controller is used and both the ea-controller and neat-controller are trained and validated. Guidance is disabled for this set of experiments. The agents are trained on 10 training environments each with a different seed for their random components, i.e. reference function and mechanical load, and validated on 2 validation environments that use a different type of reference function than the training environments. For the ea-controller and neat-controller a hyperparameter search is conducted over 16 different sets of hyperparameters that can be found in appendices C-B and C-C. Only the best results for each hyperparameter search are shown in section V.

C. Mechanical Load

For the two experiments that contain a mechanical load, a torque load profile is generated randomly that contains partwise linear transitions of torque, i.e. the load torque is interpolated linearly between two randomly chosen torque values in a random step distance. This load is refered to as uncorrelated random linear load because its values are completely uncorrelated with the state of the system itself. The load torque can be described by the following formula:

$$T_{load}[k] = \begin{cases} \frac{T_2-T_1}{k_2} \cdot k + T_1 & 0 \leq k \leq k_2 \\ \frac{T_3-T_2}{k_3-k_2} \cdot (k-k_2) + T_2 & k_2 \leq k \leq k_3 \\ \vdots & \\ \frac{T_n-T_{n-1}}{k_n-k_{n-1}} \cdot (k-k_{n-1}) + T_{n-1} & k_{n-1} \leq k \leq k_n \end{cases} \tag{11}$$

Where T_i is sampled from a uniform distribution, such that $T_i \sim \mathcal{U}(T_{min}, T_{max}) \; \forall \; i \; \in \; [1, 2, .., n]$ and k_i is sampled like $k_i \sim \mathcal{N}(d_{mean}, d_{std}) \; \forall \; i \; \in \; [1, 2, .., n]$. The load can be configured concerning the limits of the uniform distribution, which will be chosen as a fraction of the systems torque limits. Parameters d_{mean} and d_{std} describe the normal distribution that is used to sample the length of each linear transition. For this experiment the parameters are set to $T_{min} = 0.5 \cdot T_{system,min}$, $T_{max} = 0.5 \cdot T_{system,max}$, $d_{mean} = 1000$ and $d_{std} = 100$.

D. Reward function

The reward function is composed of two components: precision and electrical energy input to the system. Each term is constructed by first quantifiying the high level objective and then finalizing it using an appropriate reward finalization function.

The first term, namely precision, can be quantified very intuitively as the L2-distance between the current value and the corresponding reference value, e.g. ω and ω_{ref}:

$$L2[k \cdot \tau] = (\omega[k \cdot \tau] - \omega_{ref}[k \cdot \tau])^2 \tag{12}$$

As this term does not meet any of the three demands stated in section III-D, it is finalized using the NiLogLU function described before. The normalization borders are set according to the physical limits of the system used.

The second part of the reward function is dedicated to minimizing the energy input to the system. This term is quantified by the electrical energy input to the system:

$$E_{input}(t) = \int_0^\infty P_{el}(t)dt = \int_0^\infty i(t) \cdot u(t)dt \tag{13}$$

As the reward function has to be calculateable at each distinct time step and due to the discretization of the system into distinct time steps τ, the reward term at step k is described by:

$$E_{input}[k \cdot \tau] = i[k \cdot \tau] \cdot u[k \cdot \tau] \cdot \tau \tag{14}$$

To once again address the saturating behavior of this quantity, i.e. its optimal value being bound to 0, the NiLogLU is applied. The interval [a,b] for normalization is calculated by multiplying the limits of the physical system for current i and voltage u with the time step size τ to arrive at the maximum possible value for the energy input over one time step.

In total, the complete reward function boils down to the following:

$$r[k] = \lambda_1 \cdot NiLogLU(L2[k]) + \lambda_2 \cdot NiLogLU(E_{input}[k]) \tag{15}$$

E. Metrics

To track both optimization goals, two metrics are used for each experiment. Firstly, the average absolute deviation from reference is calculated over a 10000 step episode:

$$M_{||\omega-\omega_{ref}||} = \frac{1}{10000} \cdot \sum_{i=1}^{10000} ||st_i[\omega] - st_i[\omega_{ref}]|| \tag{16}$$

Where $st_i[\omega]$ and $st_i[\omega_{ref}]$ denote the value of ω and ω_{ref} at step st_i respectively.

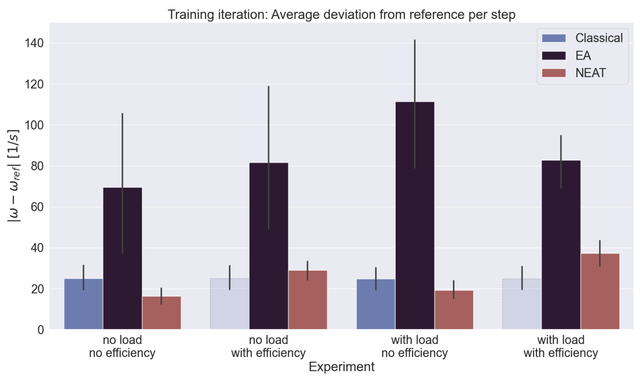
Additionally, the average electrical energy input to the system per step is tracked over the same episode of 10000 steps in the following way:

$$M_E = \frac{1}{10000} \cdot \sum_{i=1}^{10000} st_i[i] \cdot st_i[u] \cdot \tau \tag{17}$$

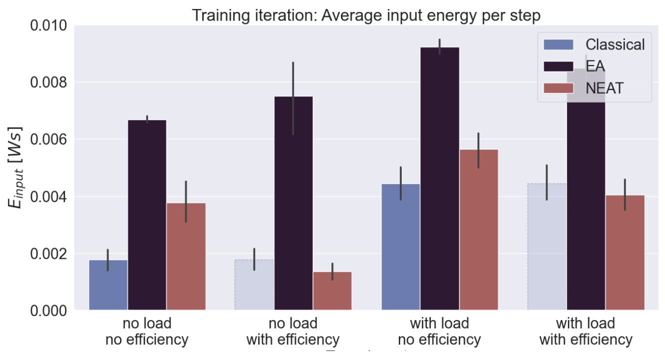
With $st_i[i]$, $st_i[u]$ being current and voltage at step st_i and τ the time step size used for the simulation.

V. Results

Figures 3a and 3b show the experiment results on the training environments. As can be seen, the classical controller and the NEAT-agent outperform the EA-agent by a factor of 2-3 over all four experiments conducted, both in terms of deviation and energy input. Adding a load torque considerably



(a)



(b)

Fig. 3: Experiment results on the training environments seperated for each agent type and experiment run. As the classical controller is the same for all four experiments, it yields the same results with or without a reward for energy efficiency. This is indicated by the slight transparency of the corresponding bars. The bars show the mean value over all 10 training environments and the grey error bars indicate the 95% confidence interval. (a) Displays the average absolute deviation from reference per step and (b) summarizes the average input energy per step.

increases energy consumption for all approaches, which is reasoned by the necessity to compensate for this induced load.

However, the most prominent feature is the ”precision vs. energy efficiency” tradeoff of the NEAT agent. When optimizing merely for precision, NEAT outperforms the classical approach by around 30% with no load and by 25% under load. This comes at the cost of a drastically increased energy consumption, doubling the amount needed for the classical controller in the no-load case and increasing it by 20% under load. When instead optimizing jointly for a low energy consumption an inverse relationship is revealed. Now, NEAT yields a superior energy consumption by 15% without and 5% with load, but becomes increasingly inaccurate, especially under load. This tradeoff introduces a new degree of freedom that is unavailable for the classical approach. Depending on the usecase, a control system engineer may choose to sacrifice some precision to enhance the energetic footprint of the controller and vice versa [1].

In order to identify the origin of this tradeoff, we analyze how the control behavior of the NEAT-agent changes when optimizing for precision alone compared to optimizing for both precision and energy efficiency jointly. The results of this investigation are shown in figures 4a to 4c. The most distinct discrepency between the three trajectories are the dynamics in terms of current i and voltage u. This discrepency is especially prominent when comparing the NEAT-agent optimized merely for precision with both other agents, as the NEAT-agent changes its action very dynamically. It does not only do so quickly, but also uses the entire dynamic range of the current i without ever actually violating the physical limit. It is this versatility, that allows the agent to adapt rapidly to changes and even small fluctuations in the reference function, thus making it the most precise solution as previously discussed in figure 3a. Combining its tendency to choose actions at the border of the available interval with the quickly varying actions, results in the higher energy consumption.

Consequently, those two attributes are adjusted when adding the reward term rewarding low energy input. Not only is the dynamic range reduced considerably, but subsequent actions also correlate with each other more than before. This less fluctuative behavior leads to the reduction of input energy, thus addressing the secondary optimization goal. As a drawback, the system is less agile and small fluctuations in the reference function are ignored, resembling a low-pass filtered version of the original reference signal.

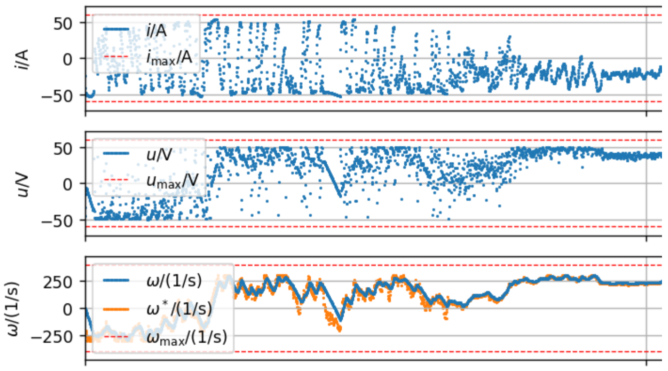
Being neither as precise as the precision-optimized NEAT-agent, nor as energy efficient as its jointly-optimized NEAT counterpart, the classical controller positions itself in between those two polarities. Therefore, it is not suprising that it contains characteristics of both other approaches, namely a dynamic range in between those of the two former, variating its actions slightly more than the energy-optimized NEAT-agent. The integrative component of the cascaded PID controller explains its more consequent tracking of the reference function in more long lasting dips or peaks, while also reasoning the apparent inability to quickly adapt to short-time fluctuations in the reference. A major drawback of this configuration is the fact that limit violations can occur, e.g. right in the lowest dip of the reference function, where the current breaches the lower current limit. A statistical assesment of the occurence rate of those limit violations can be found in figures 7a and 7b in appendix E.

An extension of this analysis, namely the network architectures representing both of the NEAT solutions discussed above, can be found in figures 8a and 8b in appendix E.

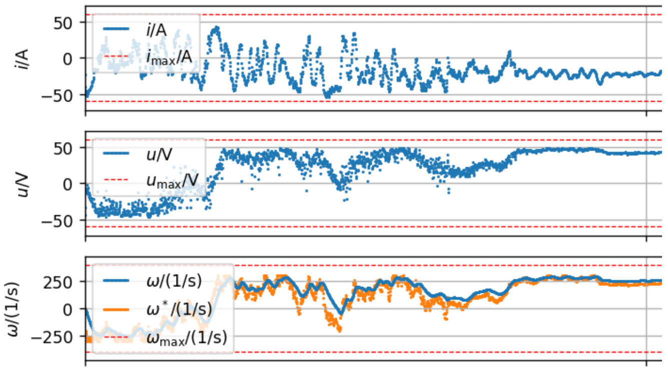
VI. Conclusion

This work has proven the applicability of NEAT to controller design and automatic controller synthesis. Along with this proof of concept, an addition to the GEM-framework was presented, that allowed to benchmark approaches of very

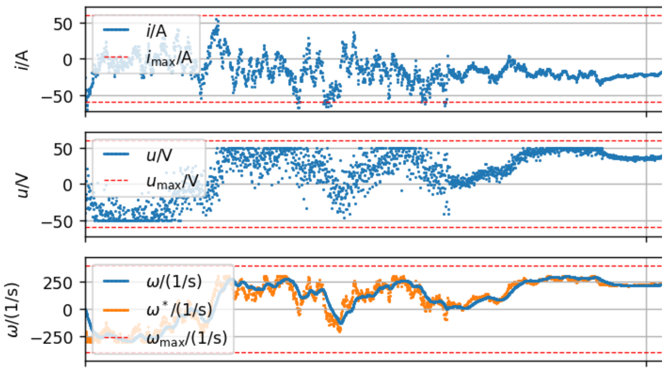
[1]The qualitative trends discussed above transfer identically to the results yielded on the validation environments shown in figures 6a and 6b in appendix E.



(a)



(b)



(c)

Fig. 4: Control behavior over the first 0.02 seconds of a play session on a training environment for (a) a NEAT-agent optimized merely for precision, (b) a NEAT-agent optimized jointly for efficiency and (c) a classical controller. For each agent, the trajectory of the current i, the voltage u and ω are shown in blue. The plot for the target variable ω also contains the trajectory of the reference value ω_{ref} in orange. Additionally, each plot contains the physical limits of the system as dashed red lines.

different underlying structure within the same framework. We identified a tradeoff between precision and lower energy consumption that is made exploitable by combining NEAT with multiobjective optimization. Using this tradeoff wisely allows to find controllers that outperform the classically derived controllers in a predefined aspect and thus breach the boundaries imposed by classical approaches. The experiments were conducted on a linear environment, thus favouring the classical approach and not playing to the strengths of NEAT, namely its non-linear solution space. Extending the comparison to non-linear environments is the subject of future research. Equally interesting is the extension of the benchmark algorithms by Actor-Critic based methods, as another representative for the NN-based approaches and fuzzy controller design, representing one of the modern control system engineering tools. To be competitive among those advanced control system design patterns further improvements to NEAT, like an improved environment exploration via guidance, may be necessary. But the potentially unlimited solution space, the applicability to multi-objective optimization and, compared to fuzzy systems and Actor-Critic methods, human-interpretable solutions, make NEAT a valid competitor in modern control system engineering.

References

[1] Praneeth Balakrishna et al. “gym-electric-motor (GEM): A Python toolbox for the simulation of electric drive systems”. In: *Journal of Open Source Software* 6.58 (2021), p. 2498. DOI: 10.21105/joss.02498. URL: https://doi.org/10.21105/joss.02498.

[2] Stuart Bennett. *A history of control engineering, 1930-1955*. 47. IET, 1993.

[3] Gerrit Book et al. “Transferring Online Reinforcement Learning for Electric Motor Control From Simulation to Real-World Experiments”. In: *IEEE Open Journal of Power Electronics* 2 (2021), pp. 187–201. DOI: 10.1109/OJPEL.2021.3065877.

[4] Bo Fan et al. “Asymptotic Tracking Controller Design for Nonlinear Systems With Guaranteed Performance”. In: *IEEE Transactions on Cybernetics* 48.7 (2018), pp. 2001–2011. DOI: 10.1109/TCYB.2017.2726039.

[5] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. “Efficient Non-linear Control Through Neuroevolution”. In: *Machine Learning: ECML 2006*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 654–662. ISBN: 978-3-540-46056-5.

[6] Sandeep Hanwate, Archit Ahuja, and Prathamesh Khare. “Design a Robust Controller for BLDC speed control”. In: *2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA*. 2022, pp. 1–6. DOI: 10 . 1109 / ICCUBEA54992.2022.10011128.

[7] M. Willjuice Iruthayarajan and S. Baskar. “Evolutionary algorithms based design of multivariable PID controller”. In: *Expert Systems with Applications* 36.5 (2009), pp. 9159–9167. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2008.12.033. URL: https://www.sciencedirect.com/science/article/pii/S0957417408008920.

[8] Carl Knospe. “PID control”. In: *IEEE Control Systems Magazine* 26.1 (2006), pp. 30–31.

[9] Xiaohang Li et al. “Nonfragile Fault-Tolerant Fuzzy Observer-Based Controller Design for Nonlinear Systems”. In: *IEEE Transactions on Fuzzy Systems* 24.6 (2016), pp. 1679–1689. DOI: 10.1109/TFUZZ.2016.2540070.

[10] Patricia Ochoa, Oscar Castillo, and José Soria. “Optimization of fuzzy controller design using a Differential Evolution algorithm with dynamic parameter adaptation based on Type-1 and Interval Type-2 fuzzy systems”. In: *Soft Computing* 24.1 (Jan. 2020), pp. 193–214. ISSN: 1433-7479. DOI: 10.1007/s00500-019-04156-3. URL: https://doi.org/10.1007/s00500-019-04156-3.

[11] Sidhartha Panda. “Multi-objective evolutionary algorithm for SSSC-based controller design”. In: *Electric Power Systems Research* 79.6 (2009), pp. 937–944. ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2008.12.004. URL: https://www.sciencedirect.com/science/article/pii/S0378779608003222.

[12] Gilberto Reynoso-Meza et al. “Multiobjective evolutionary algorithms for multivariable PI controller design”. In: *Expert Systems with Applications* 39.9 (2012), pp. 7895–7907. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2012.01.111. URL: https://www.sciencedirect.com/science/article/pii/S0957417412001297.

[13] Maximilian Schenke, Barnabas Haucke-Korber, and Oliver Wallscheid. “Finite-Set Direct Torque Control via Edge Computing-Assisted Safe Reinforcement Learning for a Permanent Magnet Synchronous Motor”. In: (Feb. 2023). DOI: 10.36227/techrxiv.22032578.v2. URL: https://www.techrxiv.org/articles/preprint/Finite-Set_Direct_Torque_Control_via_Edge_Computing-Assisted_Safe_Reinforcement_Learning_for_a_Permanent_Magnet_Synchronous_Motor/22032578.

[14] Maximilian Schenke, Wilhelm Kirchgässner, and Oliver Wallscheid. “Controller Design for Electrical Drives by Deep Reinforcement Learning: A Proof of Concept”. In: *IEEE Transactions on Industrial Informatics* 16.7 (2020), pp. 4650–4658. DOI: 10.1109/TII.2019.2948387.

[15] Kenneth O. Stanley and Risto Miikkulainen. “Efficient Evolution Of Neural Network Topologies”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Ed. by William B. Langdon et al. Piscataway, NJ: San Francisco, CA: Morgan Kaufmann, 2002, pp. 1757–1762. URL: http://nn.cs.utexas.edu/?stanley:cec02.

[16] Suryakant et al. “Minimization of torque ripples in PMSM drive using PI- resonant controller-based model predictive control”. In: *Electrical Engineering* 105.1 (Feb. 2023), pp. 207–219. ISSN: 1432-0487. DOI: 10.1007/s00202-022-01660-y. URL: https://doi.org/10.1007/s00202-022-01660-y.

[17] K.S. Tang et al. “An optimal fuzzy PID controller”. In: *IEEE Transactions on Industrial Electronics* 48.4 (2001), pp. 757–765. DOI: 10.1109/41.937407.

[18] Vinod K Valsalam et al. “Constructing controllers for physical multilegged robots using the enso neuroevolution approach”. In: *Evolutionary Intelligence* 5 (2012), pp. 45–56.

[19] Guichao Yang et al. “Output feedback adaptive RISE control for uncertain nonlinear systems”. In: *Asian Journal of Control* 25.1 (2023), pp. 433–442. DOI: https://doi.org/10.1002/asjc.2793. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asjc.2793. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/asjc.2793.

[20] Feng Zheng et al. “Robust PI controller design for nonlinear systems via fuzzy modeling approach”. In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 31.6 (2001), pp. 666–675.

APPENDIX A NOTATION

The following notations are chosen according to common literature in RL.

1) State: A state s denotes a set of variables that describe the entire system at a fixed time. Therefore, this set of variables depends on the system in question. For the permanently excited motor the state of the system is described by:

$$s = \{u_{in}, i_{sup}, i_{in}, \omega, T, reference\} \quad (18)$$

Note that the state of the system in RL is often chosen as the mathematical state of the system extended by inputs and, in the case of a control task, the reference value.

2) Step: A step st is a description of a transition from one state of the environment to another. It does not only contain the state, but also the action taken and the reward gained. Therefore it can be described as:

$$st = \{s, a, r, d, s'\} \quad (19)$$

Where s is the current state, a the action taken, r the returned reward, d a boolean flag signaling that the environment needs to be reset to be used again and s' the next state.

3) Episode: An episode E is a sequence of subsequent steps. An episode ends when the environment sets the done flag of a step, signaling the necessity of a reset. Thus an n-step Episode looks like the following:

$$E = \{st_1, st_2, ..., st_n\} \quad (20)$$

APPENDIX B EXPERIMENT PARAMETERS

The following table contains a condensed description of the parameters used in the experiments conducted within this work. A description of the Environments used can be found in the follow up section D.

APPENDIX C AGENT PARAMETERS

A. Classical-controller

The classical-controller is realized in a cascaded manner, containing an outer loop that uses the deviation $\Delta\omega = \omega_{ref} - \omega$ as an input to calculate a reference current i_{ref}. The deviation from this reference current $\Delta i = i_{ref} - i$ serves as the input to the inner loops controller that then outputs the voltage that is

TABLE I: Parameters of Experiments

Parameter	Value
Number of training iterations	250
Number of training environments	10
Number of validation envrironments	2
Number of guidance environments	0
Number of steps per training play session	200
Number of steps per validation play session	10000
Number of steps per guidance play session	10000
Normalize fitness over episode length	False
Normalize fitness over number of episodes	False
Stop training session on done	True

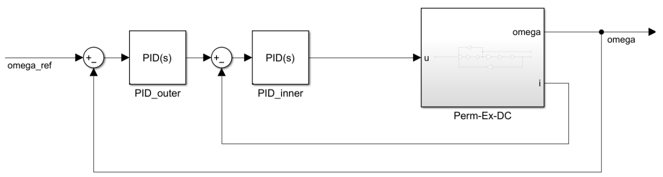


Fig. 5: Matlab Simulink model of the cascaded controller structure used for the classical controller agent. Except for the name of the inner loops control variable, the same model is used for the ea-agent.

TABLE II: Parameters of the Classical-Agent.

Parameter	Value
Kp_{inner}	1.187500
Ki_{inner}	1000.0
Kd_{inner}	0.0
Kp_{outer}	0.346331
Ki_{outer}	1000.0
Kd_{outer}	0.000005
$Windup-guard_{inner}$	100
$Windup-guard_{outer}$	100

supposed to be input to the motor. This structure is visualized in figure 5:

The controller is designed such that the inner loop compensates a slow pole of the system and ensure that there is no remaining control error. The cascaded structure is chosen, because the inner loop can then also be used to ensure that the current i is bounded within its physical limits by designing the inner loop with no overswing potential. The outer loop is then chosen accordingly to allow precise and quick control of the system as a whole. The python implementation extends the standard PID structure with a windup guard, which is the limit of the integral term in the PID controller, to counteract the problem of integral windup. The value of this term is chosen emperically. This results in the following parameters:

B. *EA-controller*

The ea-controller is used to tune the parameters of the cascaded PID-controller presented in C-A. The only additional degree of freedom added is the allowance of any available variable as the feedback variable of the inner loop. Instead of using the current i like the classical controller, the ea-controller could use the torque T for example. The choice of this variable is the only additional parameter that may be tuned by the EA. As for the NEAT-controller a hyperparameter search is done for the EA-controller. The hyperparameters subject to this search are denoted with a * in the following table that describes the EA-controllers parameters used for the experiments:

TABLE III: Hyperparameters of EA-agent. Parameters denoted with * are subject to the hyperparameter search within the given boundaries.

Parameter	Value
Population size	150
Mutation probability*	[0.1, 0.2, .., 0.9]
Elite ratio	0.05
Crossover probability*	[0.1, 0.2, ..,0.9]
Parents portion	0.3
Crossover type	uniform
Controller type	cascaded

C. *NEAT-controller*

The NEAT controller is not restricted to the predefined cascaded structure of the classical- and EA-controller. Instead the controller is substituted with a neural network with exactly one neuron in the output layer. The output of the neural network is then used as the action of the agent. As stated before, a hyperparameter search is conducted and parameters that were subject to this search are denoted with a * followed up by the set of values that were examined during the hyperparameter search. In total, the parameters used for the NEAT-controller are the following:

TABLE IV: Hyperparameters of NEAT-agent. Parameters denoted with * are subject to the hyperparameter search within the given boundaries.

Parameter	Value
Fitness criterion	max
Fitness threshold	10000
Population size	150
Reset on extinction	False
Activation function	Sigmoid
Aggregation function	Sum
Bias init mean	0.0
Bias init std	1.0
Bias max value	30.0
Bias min value	-30.0
Bias mutate power	0.5
Bias mutate rate	0.7
Bias replace rate	0.1
Compatibility disjoint coefficient	1.0
Compatibility weight coefficient	0.5
Connection add probability*	[0.1, 0.2, .., 0.9]
Connection delete probability	0.5
Connection enabled default	True
Connection enabled mutation rate	0.01
Feed forward	True
Initial connection	Full
Node add probability*	[0.1, 0.2, .., 0.9]
Node delete probability	0.2
Number of hidden layers	0
Number of input nodes	6
Number of output nodes	1
Weight init mean	0.0
Weight init std	1.0
Weight max value	30
Weight min value	-30
Weight mutate power	0.5
Weight mutate rate	0.8
Weight replace rate	0.1
Species compatibility threshold	3.0
Species fitness function	max
Max generations of stagnation	20
Species elitism	1
Elitism	2
Survival threshold	0.2
Min Species size	2

APPENDIX D
ENVIRONMENT PARAMETERS

Within the experiments the following environment is used. This environment is varied over the four different sets of experiments only in terms of reward function and mechanical load. The changes in those parameters are denoted in seperate sections within their corresponding tables, as can be seen in table VI.

TABLE V: Electric Motor parameters

Parameter	Value
GEM motor	DcPermanentlyExcitedMotor
Armature resistance R_A	$16\,\mathrm{m\Omega}$
Armature inductance L_A	$19\,\mu\mathrm{H}$
Effective excitation flux Ψ'_E	$165\,\mathrm{mWb}$
Moment of inertia of rotor J_{rotor}	$10\,\mathrm{mg/m^2}$
Moment of inertia of load J_{load}	$1\,\mathrm{mg/m^2}$
Limits Angular velocity ω	$[-400\,1/\mathrm{s}$,$400\,1/\mathrm{s}]$
Limits Torque T	$[-38\,\mathrm{Nm}$,$38\,\mathrm{Nm}]$
Limits Current i	$[-210\,\mathrm{A}$,$210\,\mathrm{A}]$
Limits Voltage u	$[-60\,\mathrm{V},\ 60\,\mathrm{V}]$

TABLE VI: Reference function parameters

Training environments	
Reference generator	WienerProcessReferenceGenerator
Referenced variable	ω
Validation environment 0	
Reference generator	SinusoidalReferenceGenerator
Referenced variable	ω
Frequency range	[30,40]
Validation environment 1	
Reference generator	DeterministicStepReferenceGenerator
Referenced variable	ω
Amplitude factor	0.5
Step k	500
Episode lengths	1000
Offset factor	0.25

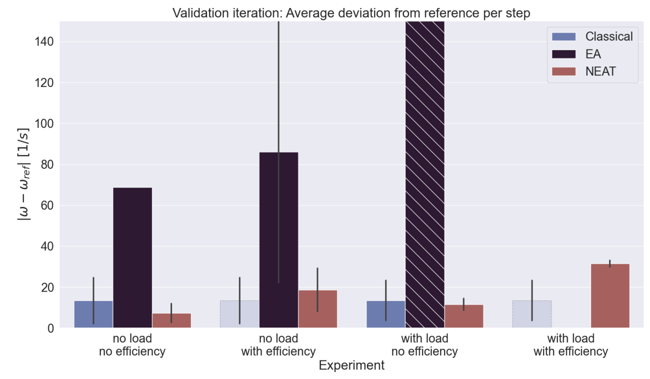
TABLE VII: Mechanical load parameters

Experiments without load	
Mechanical load	None
Experiments with load	
Mechanical load	Torque RandomLinearLoad
Episode length	10000
Transition step distance mean	1000
Transition step distance std	100
Torque amplitude factor	0.5

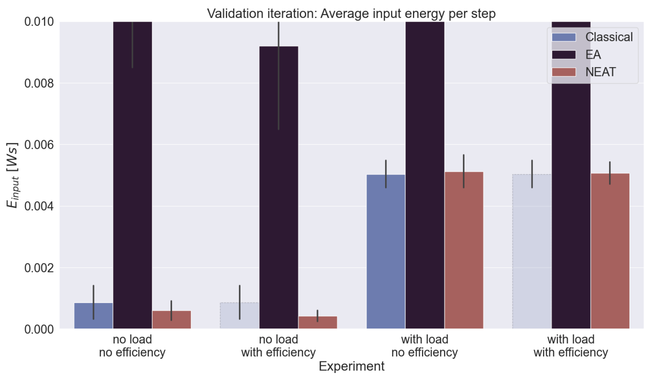
APPENDIX E
ADDITIONAL EXPERIMENT RESULTS

ACKNOWLEDGMENT

The authors would like to thank...

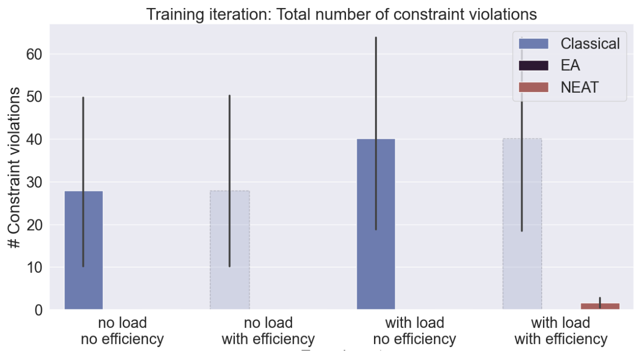


(a)

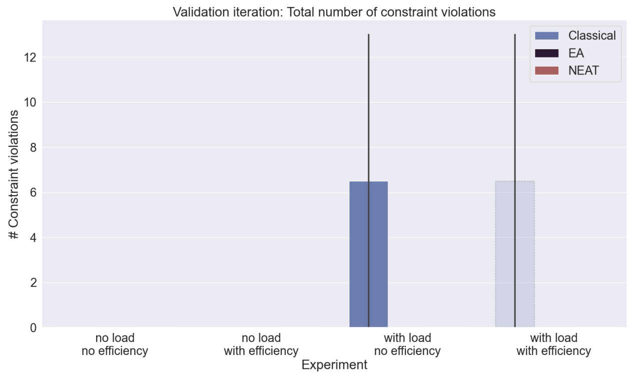


(b)

Fig. 6: Experiment results on the validation environments seperated for each agent type and experiment run. As the classical controller is the same for all four experiments, it yields the same results with or without a reward for energy efficiency. This is indicated by the slight transparency of the corresponding bars. The bars show the mean value over both validation environments and the grey error bars indicate the 95% confidence interval. (a) Displays the average absolute deviation from reference per step and (b) summarizes the average input energy per step.

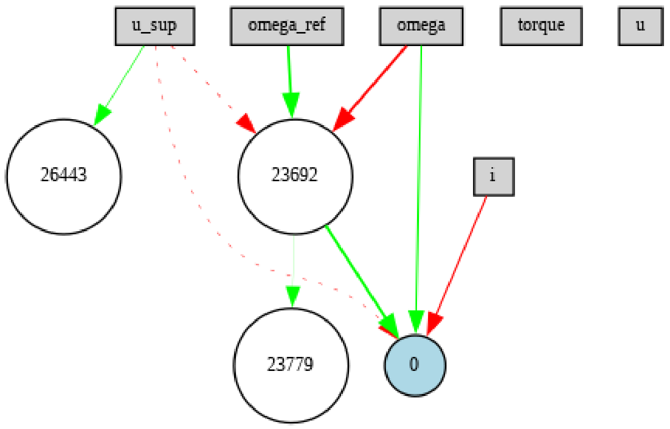


(a)

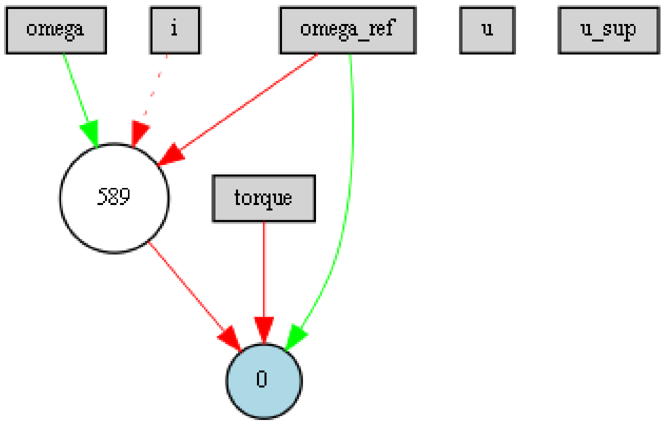


(b)

Fig. 7: Number of constraint violations for each agent type and experiment run on (a) the training environments and (b) the validation environments.



(a)



(b)

Fig. 8: Network architectures after 1000 training iterations of (a) a NEAT-agent optimized merely for presicion and (b) a NEAT-agent optimized both for precision and energy efficiency. The rectangles denote the inputs available to the network, white circles are neurons developed by NEAT containing a bias and activation function and the blue circle represents the output node of the network. The arrows express the weights of the network, where the thickness and color indicate the magnitude and sign of the corresponding weight respectively.