Machine Learning Control of a DC Electric Motor

Aidan Lambrecht, Trey Walding,
Curtis Wright
Department of Computer Science
Auburn University
Auburn, Al 36849

Abstract

The Proportional, Integral, Derivative (PID) Controller has been the prominent method of control since its development in the 1920's. Recent advancements in Machine Learning and Artificial Intelligence have introduced two additional methods for continuous state-space problems: Imitation Control and Deep Reinforcement Learning. In this paper we build upon the work of others developing imitation controllers with dense and recurrent neural network architectures. We also implement Deepmind's Deep Q Networks (DQN) algorithm. Our study confirms the performance of imitation learning control and reveals an 8.5x improvement in the response using DQN over a traditional PID controller.

1. Introduction

Proportional Integral Derivative (PID) controllers have long been used in industry tasks as feedback loops that facilitate easy control of simple machinery. Among its varied uses is its role as a DC electric motor speed controller. However, these PID controllers are not perfect: they often have problems with over-correcting the speed of the DC motor, resulting in a sinusoidal response in the motor as it accelerates past the speed that it aims for, over-corrects, and decelerates past the set point, and repeats until a balance is reached.

To correct this, a neural network controller is proposed which reduces the amount of over- and under-shoot that a PID controller creates while at the same time decreasing motor response time. In this paper, three new controllers are explored: A Deep Neural Network (DNN), a Recurrent Neural Network (RNN), and a Deep Q Network (DQN). The DQN is used to produce enormous improvements over traditional PID control systems.

2. Literature Review/Related Work

In a paper titled "On replacing PID controller with ANN controller for DC motor position control," [1] Muhammad Aamir writes about a project aimed at replacing traditional Proportional Integral Derivative control systems with Artificial Neural Networks (ANNs) as DC motor position controllers. Time delay factor and system dynamics showed that the ANN tested produced acceptable results. He even writes that "ANN control is more responsive than PID to unknown dynamics of the system," making it more versatile than traditional PIDs. Because ANNs are an outdated network architecture, we are hopeful that our own networks can improve on these old results.

In a more recent study titled "DC Motor Speed Control Using Machine Learning Algorithm," [2] Jeen Ann Abraham and Sanjeev Shrivastava write about using a Neural Network Predictive Controller (NNPC) to beat the performance of a PID tuned to control a DC motor. By simulating both in MATLAB, they were able to find that the NNPC was less accurate than the PID but did not suffer any of the overshoot caused by the PID. Additionally, when the moment of inertia of the rotor was increased by ten percent, the NNPC was able to control the motor better than the PID did, indicating the NNPC here is more adaptable than traditional PID systems.

PID controllers have various advantages and are widely used in industry. However, they suffer in long-time delay systems. This creates a problem as the parameters P, I, and D are not easy to choose. Yu,Xie, Jing, and Lu present a solution for a new control system based on a "generalized PID neural network." This solution will aid in controlling long-time delay systems. Systems like the main steam temperature control of large boiler units. These systems have the characteristics of conventional PID controllers with fixed parameters. They mention that both stability and control cannot be achieved with PID [3].

Zuo, Hu, and Li research in "PID Controller Tuning by Using Extremum Seeking Algorithm Based on Annealing Recurrent Neural Network" [4] propose a discrete-time extremum seeking algorithm that is based on annealing recurrent neural networks ESA-ARNN. This is performed by introducing a cost function and transforming the parameters into an extremum. They propose an ESA-ARNN that realizes auto-tuning for the PID controller parameters. Their research concludes with not having to rely on knowledge of the controlled plant by minimizing the cost function.

3. Infrastructure

3.1 Dataset

The dataset used to train the networks was developed specifically for this project. A DC motor and PID system was developed in Python. Data was collected and exported to a .csv file which consisted of the time step, motor speed, the set point, and the controller output voltage at that time step.

Initially data was collected as the motor accelerated to a set point of 1, 2, and 3, building up a dataset of 20k samples. However, as more data was needed to train some of the networks more data was collected for other set points as well. An example of one of the samples is shown in Table 1.

Table 1. One sample from the dataset created from the PID controller

Iteration	Speed	Set Point	Voltage
14	0.022997546	1	0.236606244

3.2 Libraries Used

This project utilized Keras and Tensorflow as a framework around which to build the ML models. SciKit-Learn, Numpy, and Pandas were also used extensively in the handling of the dataset.

3.3 Oracle/Baseline (Set-Point/PID)

For this project, both an oracle and a baseline are easy to define. The oracle (or the best possible result for the motor controller) would be an instantaneous response and acceleration of the motor to the Set Point. The baseline would be the PID controller that has been an industry standard for years. Both the oracle and baseline responses are shown in Figure 1.

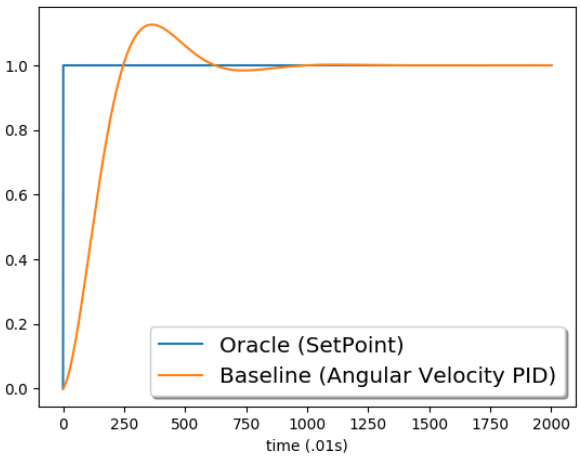

Figure 1. Oracle and baseline responses plot

The closer that the system response follows the desired setpoint, the better the controller is.

4 Experiment

4.1 DC Motor Model

For simplicity, we modeled an ideal DC electric motor where friction and motor winding efficiency were ignored. The change in angular velocity of a dc electric motor with respect to time is given by:

$$\dot{\omega} = \frac{T_s}{J}(1 - \frac{\omega}{\omega_f})$$

Where: J is the rotor moment of inertia (kg-m^2/s^2)

w is the current angular velocity (rad/s)

w_f *is* no load maximum angular velocity (rad/s)

$$\omega_f = \frac{k_t V}{k_e R}$$

T_s is the 0-speed stall torque given by:

$$T_s = \frac{k_t V}{R}$$

k_t is the motor constant (Nm/Amp)

R is resistance

V is voltage

In SI units $k_t = k_e$

4.2 PID Controller

The equation of a PID controller is given below:

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\,dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

Where: *u(t)* = The incremental control move.

e(t) = Error signal

K_p = Proportional coefficient

K_i = Integral coefficient

K_d = Derivative coefficient

4.3 Data Collection for Imitation Control

Twenty response iterations with random starting speeds between [0,5] and setpoints between [1,5] were run generating a bank of 20,000 datapoints. Each run was for 2,000, 0.01s time steps or 20 simulated seconds. Each datapoint captured the control voltage, velocity, and the setpoint.

5 Imitation Control: Dense Neural Network (DNN)

Dense Neural Networks are fully connected by the neurons within a network layer. The neurons in the current layer receive an input from the previous layer. These layers are non-linear, thus the result passes through an Activation function ex. [y = f(w*x+b)]. The drawback of a DNN is it will not detect repetition in a sequence or output different results from the same input. Therefore, this solution did not give us our optimal result, thus moving us to try RNN since it can handle these types of scenarios.

5.1 DNN Architecture

In Python with Keras, we simulated and fit a network with three hidden layers each with 14 neurons and rectified linear unit (RELU) activation function. The final output layer has a linear activation function.

5.2 DNN Conclusions

The DNN controller generates a system response that stabilizes 40% faster than the baseline PID controller, however it misses the setpoint by about 3%. It also produces a response with less overshoot. A DNN controller may be ideal for a situation where there is a prohibitive cost for overshoot, with a slight tradeoff in accuracy.

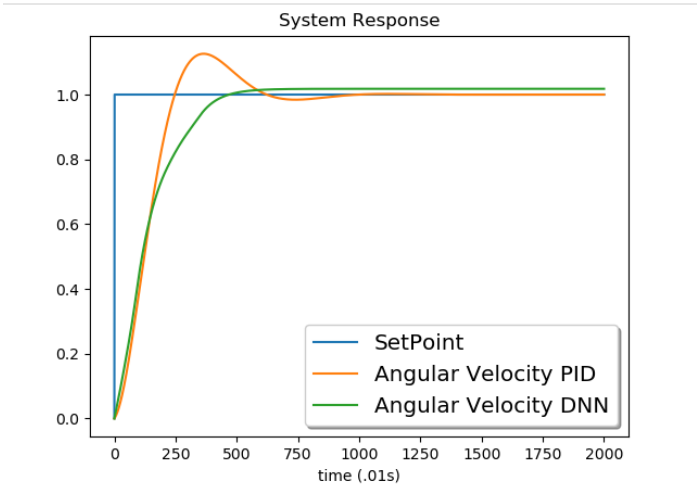

Figure 2: Dense neural network controller vs PID controller

6 Imitation Control: Recurrent Neural Network

Recurrent Neural Networks specialize in tasks where input can be written as a series. By storing the data from previous inputs, an RNN can make educated predictions based on the context of the whole input string. For instance, RNNs are easily able to predict the next number in a sequence of numbers, and they are frequently used in Natural Language Processing (NLP) tasks such as translation, text-generation, sentiment classification, and much more.

This type of network is uniquely suited to handle the task of DC motor control because it is possible to feed the network inputs that consist of several motor samples in time. By formatting input in a “Speed, Target” structure, the RNN should be able to predict the next voltage level to power the motor with precision.

6.1 RNN Architecture

Two attempts were made to implement a Recurrent Neural Network controller for the DC motor. Both architectures made use of Keras’s standard models. The first was a Keras Sequential model with one Long Short-Term Memory (LSTM) unit layer and one Dense Layer.

To preprocess data for this model, a method shown in [5] was used which takes series data with multiple features and modifies them to a form that can be used in supervised learning. For this RNN model, inputs were to be both the set point and the current speed of the motor. To facilitate better training of the model, the data was normalized to values between 0 and 1.

With the preprocessing of this model complete, the model was trained for 50 epochs with a batch size of 72. Because of the small size of the dataset, training occurred quickly.

However, due to deficient performance on the motor control task, a second RNN model was built. This model also used Keras's Sequential model as a base, but instead of an LSTM layer, this one used a Bidirectional layer. These Bidirectional RNN models use two models in conjunction with each other to output predictions.

These RNNs take input in both real-time order and reverse-time order to better understand the context of the input that they work with. To produce the final output, the outputs of these dual networks are often either summed or combined in another way [6].

In preprocessing data for this model, a simpler approach was taken. Data was still normalized to between 1 and 0, but the series data did not undergo conversion to supervised learning form. Instead, it was merely resized into inputs consisting of five-time steps each. This model trained for 1000 epochs with a batch size of 32.

6.2 RNN Conclusions

Despite seeming to be the correct network type for this task, both RNN models that were tested vastly underperformed expectations. The first network failed to even resemble a typical motor controller, increasing motor speed far past the set point for the test. This is shown in Figure 3 below.

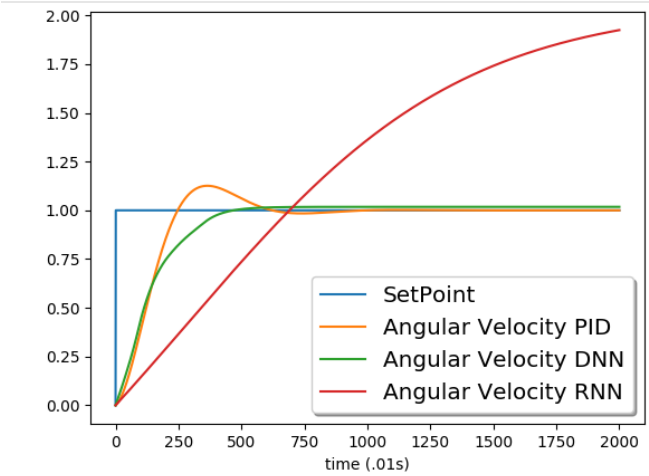

Figure 3. Response plot for DC motor with set point, PID, DNN, and first RNN controllers

The second RNN controller performed with more accuracy than the first. The results of that controller can be seen in Figure 4.

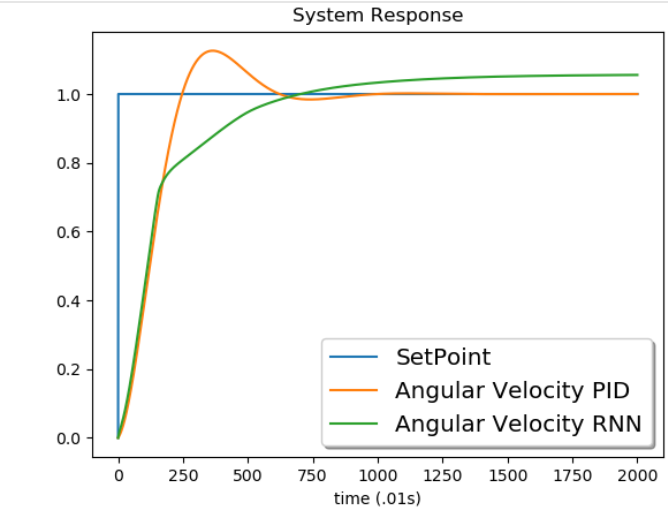

Figure 4. Response plot for DC motor with set point, PID, DNN, and second RNN controllers

The second RNN model does eventually stop accelerating the motor, though it still stabilizes at a speed that is significantly higher than the specified set-point. It results in the highest Integral Error rating out of all controllers tested, including the PID itself.

The performance of both RNN models proves them to be poor choices for a DC motor controller.

7 Deep RML Control: Vanilla Deep Q Networks

7.1 Deep Q Networks Architecture

The Vanilla DQN algorithm was implemented following the work by Deepmind [7]. Our implementation uses a replay buffer of 50,000. The Q-Network is two fully connected dense layers with 64 hidden nodes each. The inputs are the current speed, setpoint, and voltage. There are 5 outputs corresponding to the action dictionary in 7.1.1 below. The RELU activation function with a LR = 0.001was used for all layers except the last. A linear activation function was used.

Figure 4 below shows the performance of a DQN controller. Notice the system follows the setpoint closely.

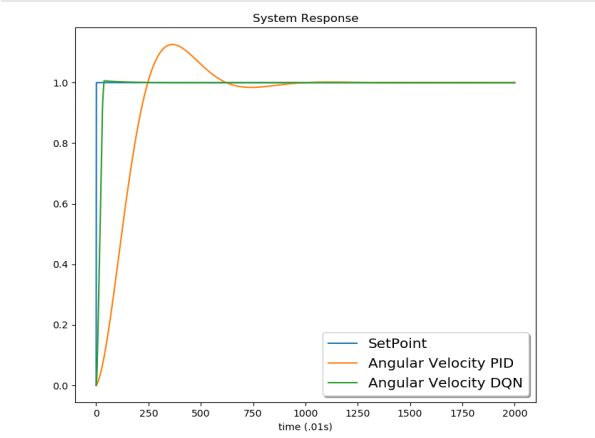

Figure 5. Response plot for the motor under DQN control vs the PID baseline

7.1.1 DQN Actions

DQN provides for a continuous state space and a discrete action space control. The choice of actions that the agent can take must be predefined. For our controller, the actions are up, down, and stay. The magnitude of these actions influences the convergence of the algorithm. Large actions result in a quick, but unstable response while small actions result in a slow but accurate response. We found the optimal set of actions to be:

$$\{0: 0, 1: 0.1, 2: -0.1, 3: 0.5, 4: -0.5\}$$

Here action 2 would subtract 0.1 from the current controller output. In the plot below, the action dictionary contained +/- 1V which the agent used to create the unstable response.

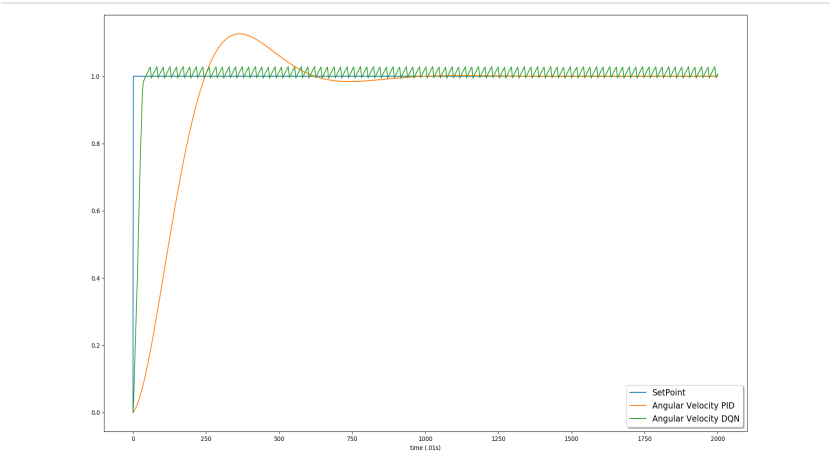

Figure 6. Response plot showing PID Vs DQN. Here the DQN algorithm has actions that are too large (+/-1V) to perfectly reach setpoint. The agent bounces back and forth with the inability to exactly reach the setpoint.

7.1.2 DQN Rewards

The ideal response for a controller is one that drives the system to the desired setpoint and remains stable thereafter. The reward scheme needs to be structured so that the agent achieves this outcome. For large state-space problems, a single reward for reaching the setpoint will often not suffice. There is not enough time for the agent to explore the entire space to find the reward before epsilon has fully decayed. This was the case for our experiment. We found the best approach was to have a positive reward at every iteration that increases as the agent approaches the setpoint, with a larger reward for stability at the setpoint. The pseudo code below shows our reward scheme.

```
Reward = 0.1 – percent error/10

If state is close to setpoint and action is stay:
        Reward += 0.1
```

We found negative rewards for incorrect actions to be ineffective. We believe this is due to the rectified linear unit (RELU) activation functions used for our Q-network. This

function truncates all negative numbers to 0. A Q-network with RELU activation where negative rewards are possible, may not be able to output accurate Q-values. If a negative reward scheme is required, we recommend using a linear or hyperbolic tangent activation function.

We found that small rewards work best. The Q-network weights are initialized between (0,1). Larger rewards result in larger Q-values. With a small learning rate and large rewards, we found it takes more iterations for the network to converge. This issue could be eliminated through gradient clipping; however, we did not explore that in this exercise.

7.2 DQN Convergence

As defined above, our algorithm converged in 180 iterations (~2 hrs. with NVIDIA RTX 2080 Super). The Convergence criteria was 25 episodes where the average time as setpoint was >80% of the episode. Each episode was 500 iterations. With a maximum reward of 0.1+0.1 = 0.2 for each time step, there is a maximum score of 100 per episode.

8 Summary of Results

The DQN algorithm can provide optimal system control, constrained by the system's physical limitations. Imitation control provides a response that is the effectively an average of responses from similar state spaces experienced during training. The plot below in figure 7 shows a comparison of all methods.

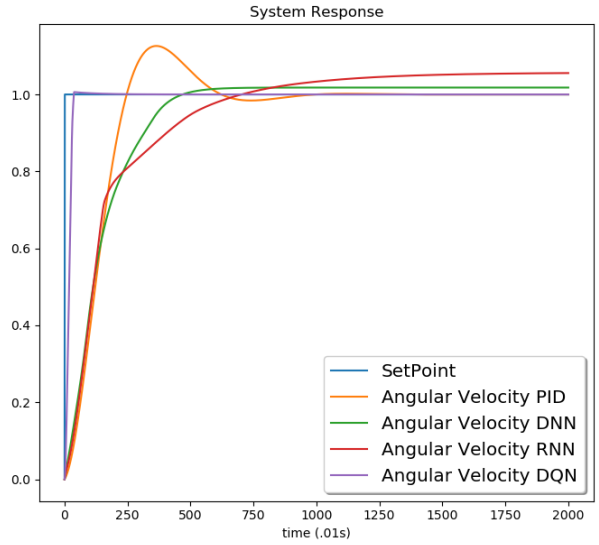

Figure 7. Response plot for DC motor with PID, Imitation (DNN, RNN) and Deep RML(DQN) controllers.

8.1 Performance Results

Each type of control system was tested for integral error, controller reversal amplitude, and controller reversal count. Results are shown in Table 2.

Table 2. Performance metrics results

	Integral Error	Controller Reversal Amplitude	Controller Reversal Count
PID (baseline)	**1.54**	**0.24**	**5***
Dense NN	**1.68**	**0.04***	**88**
Recurrent NN	**2.15**	**0.16**	**6**
DQN	**0.18***	**4.0**	**14**

*best performer for each metric

8.2 Discussion of Performance Metrics

8.2.1 Integral Error or Error Sum is the integral of the response with respect to the setpoint. This is a primary measure of control system performance. It measures both the speed and the accuracy of the response. The integral error using a DQN controller is an 8.5x improvement over the baseline PID controller.

8.2.2 Controller Reversal Amplitude measures the maximum controller move with respect to the steady state controller location. This is a measure of the system stability but can also indicate how aggressively the system responds. The dense neural network controller outperforms all other methods in this study.

8.2.3 Controller Reversal Count is the number of times the controller output changes direction. It is a measure of system stability and efficiency. Here the PID controller outperforms all other methods. The dense neural network suffers from many small controller reversals.

8.3 Conclusions

Imitation control is effective, but it allows for some drift. Its main strength is a stable response indicated by the Reversal Amplitude metric where the Dense NN and RNN controller outperformed. However, this method comes with a tradeoff in accuracy. Deep Reinforcement learning is a powerful suite of algorithms that can be used to optimize any aspect of a control system given an accurate dynamics model. When it is desirable to reach setpoint with stability as quickly as possible, DQN performs better than theoretically possible with a PID controller. Deep reinforcement learning will push the physical limitations of the hardware to maximize a reward.

9 Arguments

Classical control theory is firmly rooted in mathematics, primarily differential equations. There are robust methods for predetermining the systems response and stability when a PID controller is used. Machine learning control offers no means to predict either. Another drawback is the large data requirements that may not be available. Reinforcement machine learning models require a complete and accurate system model, which is often hard to develop. Even with a model, there is no guarantee the agent has explored the entire state space. There is no confirmation that the Q network has converged on a stable action for every possible state.

Additionally, If the system operates outside of the range of available training data, the results can be unpredictable as seen in Figure 8 below.

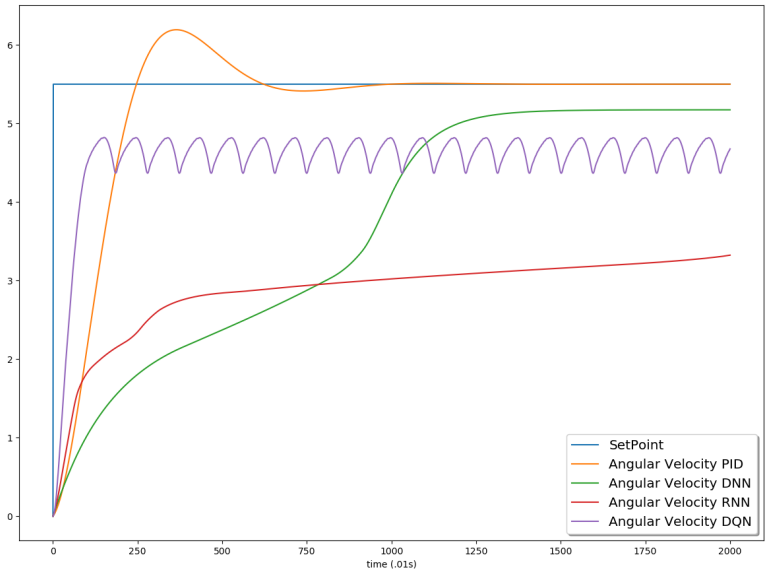

Figure 8. At setpoint of 5.5 rad/s, which is 0.5rad/s or 10% outside the range of training data, all machine learning methods fail to control the motor, while the PID controller performs as expected.

10 Conclusion

In this work, exploration was conducted into finding a new, more accurate controller for a DC motor, something previously accomplished via PID. After investigating DNN, RNN, and DQN control models, it was found that while DNN and RNN models performed worse than the baseline PID, the DQN model was performed eight times better, providing a response that is far beyond what a PID control could achieve.

Works Cited

[1] Jean Ann Abraham and Sanjeev Shrivastava, *DC Motor Speed Control Using Machine Learning Algorithm,* International Journal of Engineering Research & Technology (IJERT)ISSN: 2278-0181, April 2018.

[2] Amir Muhammad, *On replacing PID controller with ANN controller for DC motor position control,* International Journal of Research Studies in Computing April 2013.

[3] Zhun Yu, Ying-Bai Xie, You-Yin Jing, Xu-Ao Lu, *APPLYING NEURAL NETWORKS TO PID CONTROLLERS FOR TIME-DELAY SYSTEMS,* Fifth International Conference on Machine Learning and Cybernetics, Dalian, 13-16 August 2006.

[4] Bin Zuo and Yun-an Hu, *Modeling PID Controller Tuning by Using Extremum Seeking Algorithm Based on Annealing Recurrent Neural Network,* 2010 3rd International Symposium on Knowledge Acquisition and Modeling.

[5] Brownlee, Jason. *How to Convert a Time Series to a Supervised Learning Problem in Python*. Machine Learning Mastery, May 8, 2017. https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

[6] Lee, Ceshine. *Understanding Bidirectional RNN in PyTorch*. Towards Data Science, Nov. 12, 2017. https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66.

[7] Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* 518**,** 529–533 (2015). https://doi.org/10.1038/nature14236.