Journal of *Marine Science and Engineering*

MDPI

Article

Discerning Discretization for Unmanned Underwater Vehicles DC Motor Control

Jovan Menezes [1] and Timothy Sands [2,*]

[1] Department of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA
[2] Department of Mechanical Engineering (CVN), Columbia University, New York, NY 10027, USA
* Correspondence: ts2543@caa.columbia.edu

Abstract: Discretization is the process of converting a continuous function or model or equation into discrete steps. In this work, learning and adaptive techniques are implemented to control DC motors that are used for actuating control surfaces of unmanned underwater vehicles. Adaptive control is a strategy wherein the controller is designed to adapt the system with parameters that vary or are uncertain. Parameter estimation is the process of computing the parameters of a system using a model and measured data. Adaptive methods have been used in conjunction with different parameter estimation techniques. As opposed to the ubiquitous stochastic artificial intelligence approaches, very recently proposed deterministic artificial intelligence, a learning-based approach that uses the physics-defined process dynamics, is also applied to control the output of the DC motor to track a specified trajectory. This work goes further to evaluate the performance of the adaptive and learning techniques based on different discretization methods. The results are evaluated based on the absolute error mean between the output and the reference trajectory and the standard deviation of the error. The first-order hold method of discretization and surprisingly large sample time of seven-tenths of a second yields greater than sixty percent improvement over the results presented in the prequel literature.

Keywords: discretization; DC motors; deterministic artificial intelligence; adaptive control; learning control; proportional derivative; estimation; least squares; modeling

Citation: Menezes, J.; Sands, T. Discerning Discretization for Unmanned Underwater Vehicles DC Motor Control. *J. Mar. Sci. Eng.* **2023**, *11*, 436. https://doi.org/10.3390/jmse11020436

Academic Editor: Rafael Morales

Received: 12 January 2023
Revised: 13 February 2023
Accepted: 14 February 2023
Published: 16 February 2023

1. Introduction

Direct current (DC) motors, shown in Figures 1 and 2, are a class of rotating electrical motors that convert DC electrical energy into mechanical energy, and such motors have ubiquitous applications including unmanned underwater vehicles. The operation of DC motors is amidst a revolutionary change with the adoption of sophisticated microcontrollers and control strategies. Control of DC motors is a well-studied topic in the literature, including using neural networks [1,2] including neural network-based auto-tuning of classical proportional, integral, derivative controllers [3], and recursive least squares [4]. Estimators and estimation techniques are deployed side-by-side with control strategies to determine the parameters and even the state of the system using a model. Classical approaches adopted for the control of DC motors include (but are not limited to) the proportional, integral, derivative, proportional–integral, proportional–derivative, proportional–integral–derivative controller, etc. In this work, the proportional–derivative control approach is implemented, which operates on both the current and predicted process conditions. The proportional–derivative control technique provides a combination of feedforward and feedback control making it a more appropriate strategy for the learning-based approach discussed and implemented in this work.

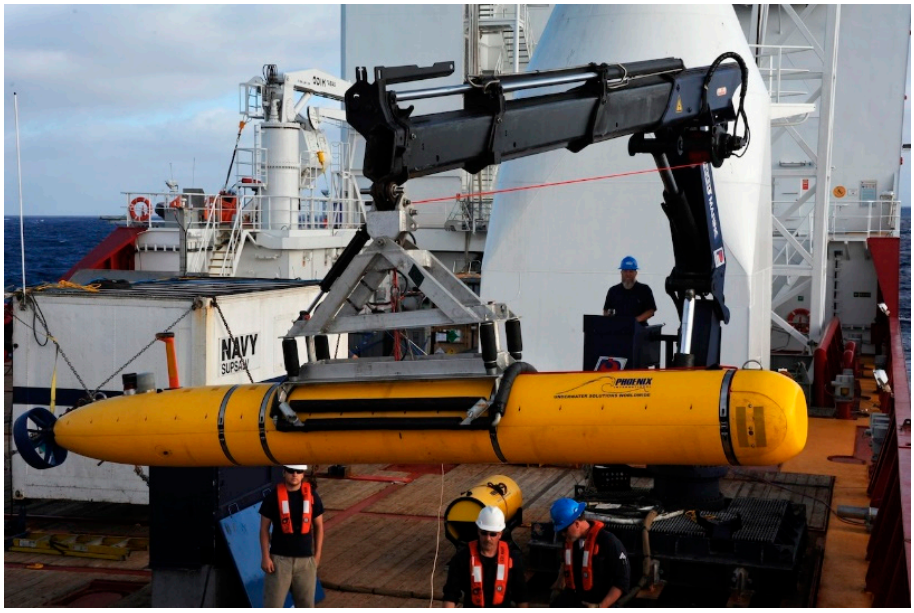


Figure 1. The propeller and control surfaces of unmanned underwater vehicles [5] generally utilize DC motors [6] like those depicted in Figure 2. Credit: Navy photo by Petty Officer 1st Class Peter D. Blair. Department of Defense photographs and imagery, unless otherwise noted, are in the public domain [7].

(**a**) (**b**) (**c**)

Figure 2. (**a**) High-torque brushless DC motors used in unmanned underwater vehicles (image credit Unmanned Systems Technology [6]). (**b**) Underwater thruster propeller motor (image credit Maxon [8]). (**c**) ECI-40 Maxon underwater drive motor and gear head (image credit Maxon [8]).

1.1. Research Lineage from the M.I.T. Rule to Regression

The method of least squares is one of the most foundational mathematical techniques used in modeling and estimation theory. The objective of the least squares method consists of adjusting the parameters of a model of the system to best fit a set of data. Estimators designed for several adaptive modeling techniques, presented in literature, are developed using the least squares approach presented in the literature [9–14], along two lines of thought represented by Slotine [9,10] as modified by Fossen [11] and Åström [12–14], respectively. Each method involves formulation of canonical regression forms, while the Slotine/Fossen approach seeks to utilize the full regression form, and the Åström approach bifurcates the regression model into components exhibiting disparate characteristics that might or might not need to be adapted. The Slotine/Fossen approach was augmented [15] with physics-based methods of Lorenz [16] to formulate the burgeoning method referred to as deterministic artificial intelligence [17], which was proposed for applications to DC motors [18] and validated by Shah [19]. Shah's validation highlighted the criticality

on motor performance of the discretization method and discretization time interval and recommended study of such. This manuscript is one such study as recommended by Shah.

A main theme of the research lineage is replacement of classical adaption methods (e.g., the co-called "M.I.T. rule") [20] with estimation methods based on least squares. Numerous variations in the least squares approach have been developed that have been used in designing different types of estimators. It is therefore worth obtaining an essence of these forms that are prevalent to the work presented in this manuscript.

1.2. Least Squares Variations

Recursive form (an adaptive algorithm that recursively estimates the parameters of a system using a model that is linear in those parameters) [14] and batch form (where all measurements are collected together and processed simultaneously) [10] are variations in the least squares approach that have also been applied as an adaptive method [15].

Another version of the method of least squares is the weighted least squares, also known as weighted linear regression, in which weights are assigned to the observations, and these weights are proportional to the reciprocal of the error variance for that observation. Ideally, the weights in the weighted least squares analysis are nonrandom quantities that are proportional to the reciprocal of the variances of the measured data, but it might not always be clear as to how to choose the weights.

In cases where there is an ambiguity in choosing the weights for the weighted least squares approach, the extended least squares method may be adopted. The extended least squares approach provides a potential solution to the weighting problem experienced in the weighted least squares method by avoiding it. The extended least squares method is a maximum likelihood kind of statistical estimation method when the data are normally distributed. With the extended least squares, weights need not be chosen.

A common modeling assumption used in this work is the autoregressive moving average model. By applying the analysis of autoregression and moving average simultaneously to time-stamped data, the autoregressive moving average method is obtained. In the autoregressive approach, the output variable is linearly dependent on its previous values as well as on a stochastic term (an imperfectly predictable term), resulting in a model in the stochastic difference equation form. The moving-average model, or the moving-average process, is a method implemented for modeling time series data that is univariate (a function involving only one variable) in nature. In the moving-average model, contrary to the autoregressive analysis, the output variable is cross-correlated with a random variable nonidentical to itself. The autoregressive moving average approach assumes that the time series, for the model being implemented, is stationary. Further, if the time series fluctuates, it is assumed to be uniformly fluctuating around a particular time.

1.3. Physics-Based Utilization of Governing Differential Equations

Various techniques, based on adaptation, have been implemented to control different types of systems with unknown parameters and achieve the desired response, which often would be an output signal tracking an input signal [21]. In most natural processes, a change in the input does not lead to a matched change in the output. By incorporating physics-based procedures, however, certain adaptive methods allow for control and tracking. The primary difference between nonlinear adaptive control [14] and the physics-based method [16] is the implementation of complete mathematical expression for modeling by physics. The feedforward portion of recent research in deterministic artificial intelligence [18] (also called assertion of self-awareness) embodies the core idea of the two techniques. Parameter adaption by classical methods (e.g., M.I.T. rule) is one option [22], while Smeresky and Rizzo propose optimal learning [23] as another option utilizing batch least squares, where current research investigates the efficacies of variations in least squares. Last year, Zhai studied learning implementation by signal-encoded deep learning [24] and offered a direct comparison to deterministic algorithms [25].

Deterministic artificial intelligence, and likewise other learning-based techniques, uses process dynamics (dictated by physics-based mathematical models) as self-awareness statements in the form of feedforward controls [18]. Learning is driven by the evaluation of performance metrics to track command-input. The novel idea in deterministic artificial intelligence is that the self-awareness statements only function when a prior desired trajectory derived analytically is provided, where the error calculation enables both adaption and optimal learning according to recently published results [19]. The latest literature shows that the adaptive approach achieves around 29% lower error than deterministic artificial intelligence in input tracking [19].

1.4. Discretization

The models developed for estimating, adapting, and learning physical systems begin as continuous functions, since the modeling is strictly taken from the first principles of physics. However, when managing controllers and computers to implement the control strategies, it is essential to discretize the continuous system. Different discretization strategies are available to convert continuous systems into discrete systems, particularly in the fields of signal processing, control, and estimation. This manuscript presents a study of the effects of different discretization methods when converting the continuous model of the DC motor and the eventual efficacy applied to DC motor control. Arbitrary selection of different discretization methods and intervals led to the results depicted in Figure 3.

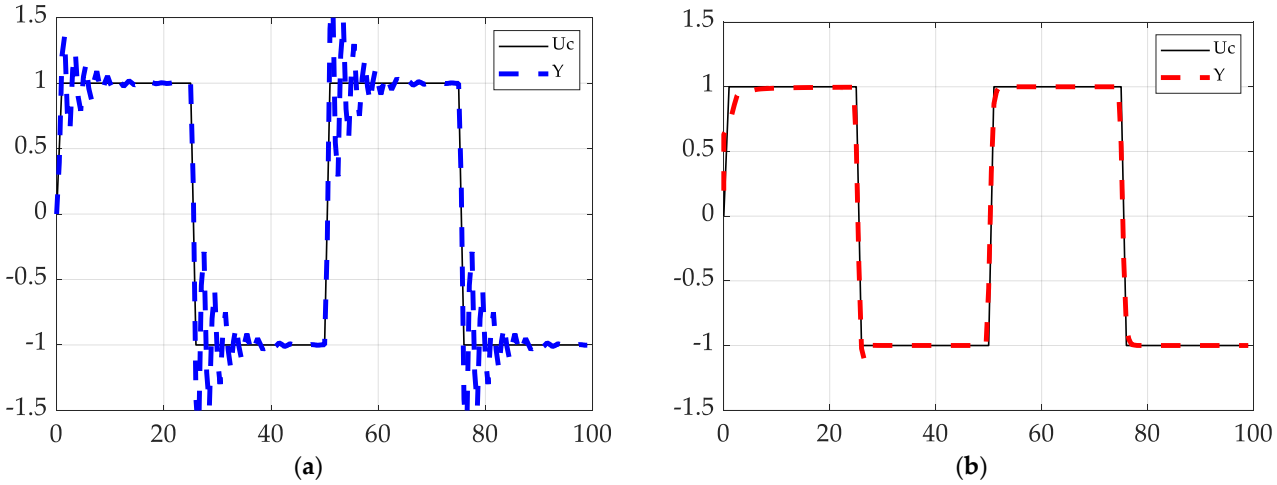


Figure 3. Recently proposed DC motor improvements described in the literature, where the difference is the discretization method and sample time [18]. The command signal is indicated by the black solid line. The output obtained using deterministic artificial intelligence is represented by the blue dotted line in (**a**). The output signal obtained through the model-following method with recursive least squares estimation is described by the red dotted line in (**b**).

This work includes a study of the effects of changing the sample time, a scalar value that represents the sampling period for the resulting discrete-time system. Discussions are offered on the efficacy of DC motors' output tracking a desired trajectory (in this work, a series of alternating step functions) with iterations of the discretization method and sample time. Finally, novel conclusions and recommendations are offered regarding what discretization method and sample time are best suited (including limiting cases) for the deterministic artificial intelligence method, based on the mean absolute error and standard deviation of the error in the output from the desired trajectory.

Main conclusion of the prequel study. *The potency of the deterministic artificial intelligence approach is limited by sample time values and discretization method. Identification of the limiting discretization values and recommendations for the discretization method is recommended.*

To summarize, the different strategies that were studied in this research are recursive least squares, extended least squares, autoregressive moving average, and deterministic artificial intelligence.

Main conclusion of this study. *The deterministic artificial intelligence approach is suitable only for a range of sample time values for each discretization method. To achieve high efficacy, these values and methods must be adhered to.*

2. Materials and Methods

To illustrate the main conclusions of this work, the Materials and Methods begin with overarching principles such as methodological process flow (e.g., Figure 4) and topology of eventual computer simulations (e.g., Figure 5). Next, Section 2.1 describes motor dynamics modeling and control strategies. Next, discretization is discussed regarding the accompanying simulation. The simulation experiments are summarized in succinct tables of common figures of merit (e.g., means and standard deviations) with accompanying figures to provide qualitative depiction and comparison of the quantitative results. The comparison leads to recommendations for the discretization method and time interval to achieve efficacy in controlling DC motors.

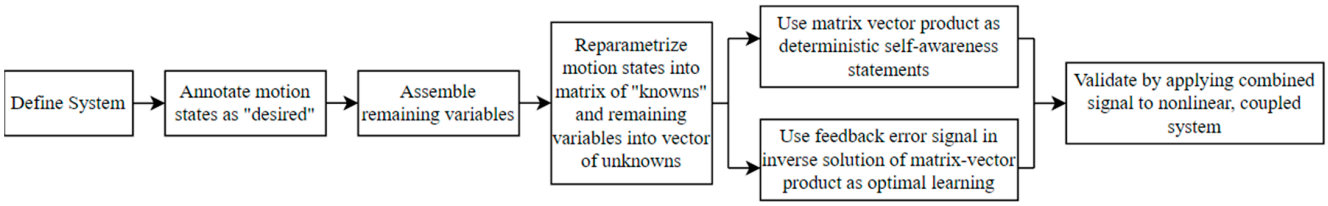


Figure 4. Topology of deterministic artificial intelligence [18] that is applied to an unmanned underwater vehicle system, as shown in Figure 5, whose actuators are powered by DC motors.

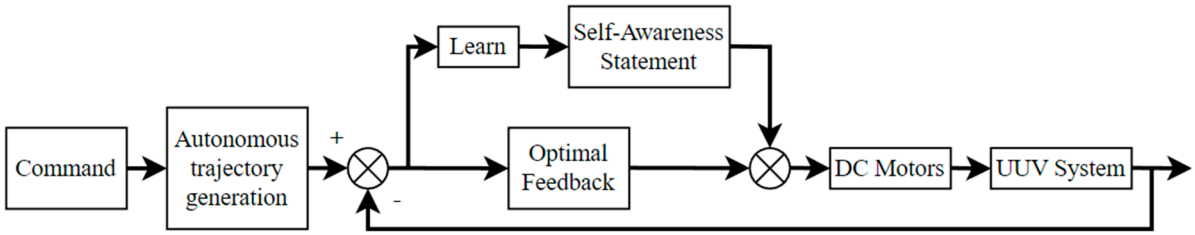


Figure 5. Topology of deterministic artificial intelligence [18] applied to unmanned underwater vehicles that are actuated using DC motors.

2.1. Modeling DC Motor Dynamics and Designing Control Strategies

The transfer function (a function in the frequency domain that relates the output of the system to its input), also known as the process equation or the process truth model, in the continuous time domain for modeling the dynamics of a DC motor is given by Equation (1). Using analytical approaches presented in [15], the discrete transfer function for the DC motor (typical specifications shown in Appendix B) is given using Equation (2):

$$G(s) = \frac{B(s)}{A(s)} = \frac{1}{s(s+1)} \tag{1}$$

$$G(z) = \frac{B(z)}{A(z)} = \frac{b_0 z + b_1}{z^2 + a_1 z + a_2} = \frac{0.1065z + 0.0902}{z^2 - 1.9z + 0.88} \tag{2}$$

The canonical motor model from [26] with bump-test current regulation is integrated for voltage display, where the bump-test reasonably resembles each discontinuous jump of the square wave. The canonical model is available for purchase by readers seeking to repeat the work in this manuscript, and its designation is Quanser USB QICii, and the

canonical motor is Equation 2.6 in Section 2.6.1.1 of the laboratory workbook, DC Motor Control Trainer (DCMCT) [26]. Module Description, a section in the workbook, elaborates the modeling methods for rate output. The author of the workbook is the same author of references [12–14], which reference the same motor model as that described herein; these references provide the benchmark methods for this manuscript, e.g., provides the starting point. This is important to allow researchers to duplicate these results.

By altering the discretization method and the sample time, the values of b_0, b_1, a_1, and a_2 in Equation (2) change, resulting in a different discrete transfer function each time. The transfer function in Equation (2) has poles at $z = 1.1,\ 0.80$ and zero at $z = -0.8469$. Thus, the system is unstable since the magnitude of one pole is greater than 1.0 (i.e., the pole lies outside the unit circle). By applying adaptive control techniques, these poles are relocated such that the system becomes stable. The deterministic artificial intelligence modeling strategy, however, implements a proportional–derivative (PD) feedback adaptive technique to follow the target path instead of attempting pole relocation. While modeling, nonwhite (correlated) noise as two delayed noise terms, having a Gaussian distribution $N(0,\ 1/625)$, are added. MATLAB® was used to perform simulations of both approaches. Appendix A provides the code used to obtain the latter mentioned results. The input signal to the learning and the adaptive control technique is an identical square wave. The former design creates trajectories that are sinusoidal in nature and begin at the initial discontinuity of the square wave while terminating at the peak of each square wave discontinuity through an autonomous path planning algorithm. Using Equation (2), the output equation for the DC motor system is obtained and given by Equation (3):

$$y(k+2) = b_0 u(k+1) + b_1 u(k) - a_1 y(k+1) - a_2 y(k) \tag{3}$$

While implementing the deterministic artificial intelligence approach, drastic forced changes are not caused in the output signal of the system to match the changes in the input signal. In contrast, for any change in the input state, the output progressively follows a calculated trajectory towards the target state in a manner such that for any time step there is not an analytically undefined position. Self-awareness is achieved by asserting process dynamics through the control mechanism in a feedforward manner, while the feedback parameters that are learned via a proportional–derivative feedback mechanism (or 2-norm optimal methods) enable modulating the control signal. The process flow of deterministic artificial intelligence is illustrated in Figure 4 while Figure 5 shows the topology and the self-awareness statements. Equation (4) demonstrates how the feedback parameters are calculated through batch least squares [23], where $\hat{\theta}$ represents the learned parameters to adjust the control input u, ϕ_d represents a matrix comprising the states of the desired trajectory, and δu is the error in the control input.

$$u = \phi_d \hat{\theta} = \phi_d \left({\phi_d}^T \phi_d \right)^{-1} {\phi_d}^T \delta u \tag{4}$$

2.2. Discretization Methods

The different discretization methods used in this study are least squares, zero-order hold, Tustin approximation, first-order hold, zero-pole matching equivalents, and impulse-invariant mapping. When the input has a staircase form and the continuous and discrete models of the system in the time domain need to be matched exactly, the zero-order hold technique of discretization is implemented. The zero-order hold discretization gives the discretized transfer function $H_d(z)$ of a continuous time linear model $H(s)$. In the zero-order hold technique, by holding every sample value $u(k)$ constant during one sample period, i.e., $u(t) = u(k) \forall\ kT_s \leq t \leq (k+1)T_s$, the continuous time input signal $u(t)$ is generated. This signal $u(t)$ serves as an input to the continuous system $H(s)$. By sampling the output of the continuous time system $y(t)$ every T_s seconds, the output for the discrete time system $y(k)$ is obtained.

When the input has a piecewise linear form and the continuous and discrete models in the time domain need to be matched exactly, the first-order hold technique of discretization is implemented. The first-order hold technique differs from the previously mentioned zero-order hold approach by the basic holding mechanism. To convert the discrete input samples $u(k)$ into a continuous input $u(t)$, the first-order hold uses linear interpolation between samples, as given by Equation (5):

$$u(t) = u(k) + (t - k * T_s)(u(k+1) - u(k))/T_s \; \forall \; kT_s \leq t \leq (k+1)T_s \tag{5}$$

In general, the first-order hold method is more accurate than the zero-order hold approach, particularly when the systems are driven by smooth inputs. The first-order hold method is more appropriately called the triangle approximation [25] since it varies from the standard causal first-order hold approach. The first-order hold method is also known as the ramp-invariant approximation. When the impulse response of the discrete time model and the corresponding continuous time model is required to be the same, the impulse-invariant mapping approach is applied. The sum of shifted copies of the continuous time model frequency response provides the related discrete-time model frequency response. The former model's impulse response $h_c(t)$ is sampled with the sampling period T to obtain the latter model's impulse response $h(k)$ such that $h(k) = \mathrm{T} * h_c(kT)$.

The Tustin approximation, also known as the bilinear approximation, provides the best match in the frequency domain between the discrete time model of the system and the corresponding continuous time model. The Tustin approach relates the transfer functions in the s-domain and z-domain through the approximation given by Equation (6):

$$z = e^{sT_s} \approx (1 + (0.5T_s)s)/(1 - (0.5T_s)s) \tag{6}$$

$$s' = (2z - 1)/(T_s z + 1) \tag{7}$$

In continuous to discrete conversions using the Tustin approach, the discretization $H_d(z)$ of $H(s)$, the continuous transfer function, is performed such that $H_d(z) = H(s')$, where s' is given by Equation (7). The states of s system are not preserved when the Tustin method is implemented to convert a state-space model. The state-space matrices and the time delays in the system play a key role in determining the state transformation. When the system has poles at $z = -1$, this method is not defined, while it is ill-conditioned for those with poles near $z = -1$. This method of approximation is used when good matching is required between the discrete and the continuous time models in the frequency domain. The Tustin approach is also the best suited discretization method when the model of the system has important dynamics at certain frequency that need to be captured.

The zero-pole matching equivalents conversion method applies only to systems with single-input, single-output. The discretized and continuous models have matching DC gains. The zeros and poles of the continuous and discretized system are related by the transformation $z_i = e^{s_i T_s}$, where s_i represents the ith zero or pole of the continuous time model, z_i represents the ith zero or pole of the discrete time model, and T_s represents the discretization sample time. The zero-pole matching equivalents approach is preferred when the system model at hand has a single input and a single output, and good matching is desired in the frequency domain between the discrete and continuous time models.

The least squares method of discretization implements a vector-fitting optimization strategy in order to reduce the error between the discrete and continuous time model frequency responses of the system up to the Nyquist frequency. The Nyquist frequency (or folding frequency) is a characteristic of the operation that extracts samples from a continuous time signal (also known as a sampler) and converts it to a discrete sequence. The value of the Nyquist frequency is one-half of the sampling rate. The obtained discrete time sequence is free of distortion if the highest frequency of the signal (bandwidth) is less than the Nyquist frequency. The least squares method is useful to capture fast system dynamics and large sample times are desired, e.g., if the computational resources are limited. The least squares method is only suitable for changing continuous to discrete

systems and for single-input, single-output systems. Like the zero-pole matching and the Tustin approximation, the least squares method results in a good match between the original continuous model's and the converted discrete model's frequency responses. However, for the same sample time, the least squares approach leads to a smaller difference in the discrete and continuous frequency responses when compared with the zero-pole matching or Tustin approximation. A slower sample time results in a reduced load on the processor, which is beneficial when there are limitations on the computational resources.

3. Results

Simulations were performed for different time samples using each discretization method. Results are shown qualitatively in Figure 6 with corresponding description in Table 1. Each line plotted in Figure 6 represents one of the four control strategies designed for the DC motor system. Section 4 provides the quantitative results corresponding to the depicted qualitative results.

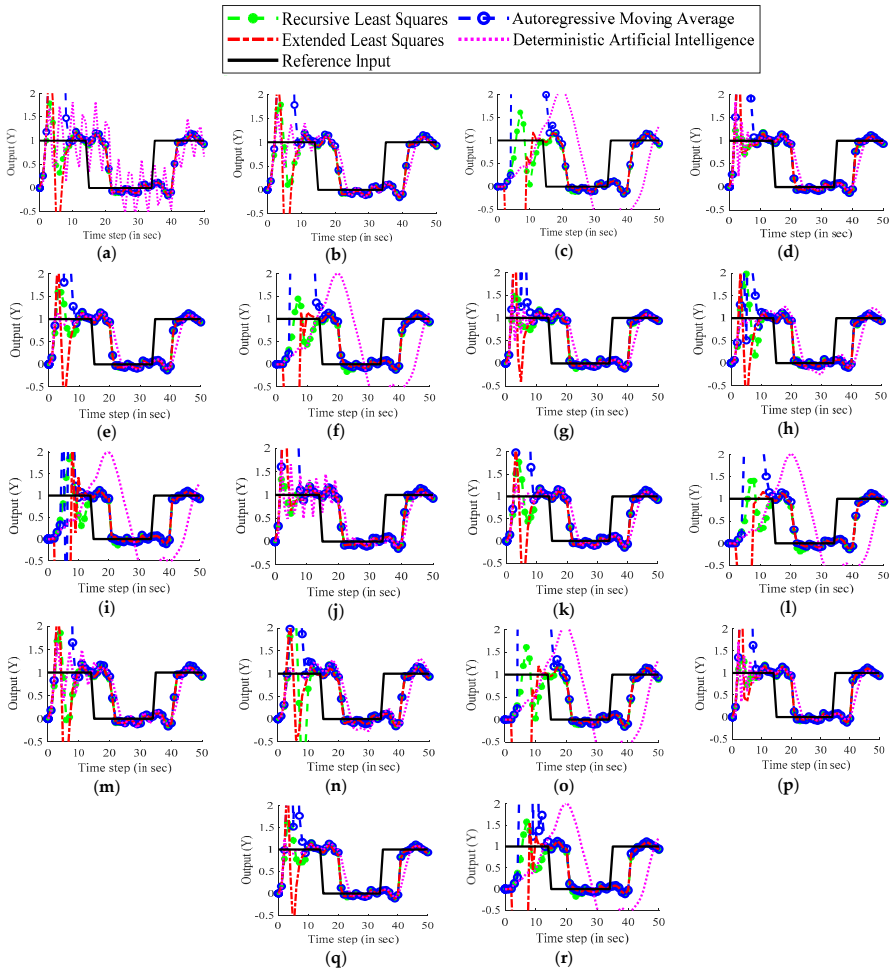


Figure 6. Output plot for each of the four control strategies with different discretization methods and sample time. (**a**) zero-order hold with 0.6 s sample time, (**b**) zero-order hold with 0.5 s sample time, (**c**) zero-order hold with 0.1 s sample time, (**d**) first-order hold with 0.7 s sample time, (**e**) first-order hold with 0.5 s sample time, (**f**) first-order hold with 0.1 s sample time, (**g**) impulse-invariant mapping with 0.5 s sample time, (**h**) impulse-invariant mapping with 0.3 s sample time, (**i**) impulse-invariant mapping with 0.1 s sample time, (**j**) Tustin approximation with 0.8 s sample time, (**k**) Tustin approximation with 0.5 s sample time, (**l**) Tustin approximation with 0.1 s sample time, (**m**) zero-pole matching with 0.5 s sample time, (**n**) zero-pole matching with 0.3 s sample time, (**o**) zero-pole matching with 0.1 s sample time, (**p**) least squares with 0.6 s sample time, (**q**) least squares with 0.5 s sample time, (**r**) least squares with 0.1 s sample time.

Table 1. Discretization method and sample time used for each plot displayed in Figure 6.[1]

Plot	Description	Plot	Description	Plot	Description
(a)	ZOH and $T_s = 0.6\ s$	(b)	ZOH and $T_s = 0.5\ s$	(c)	ZOH and $T_s = 0.1\ s$
(d)	FOH and $T_s = 0.7\ s$	(e)	FOH and $T_s = 0.5\ s$	(f)	FOH and $T_s = 0.1\ s$
(g)	IIM and $T_s = 0.5\ s$	(h)	IIM and $T_s = 0.3\ s$	(i)	IIM and $T_s = 0.1\ s$
(j)	TA and $T_s = 0.8\ s$	(k)	TA and $T_s = 0.5\ s$	(l)	TA and $T_s = 0.1\ s$
(m)	ZPM and $T_s = 0.5\ s$	(n)	ZPM and $T_s = 0.3\ s$	(o)	ZPM and $T_s = 0.1\ s$
(p)	LS and $T_s = 0.6\ s$	(q)	LS and $T_s = 0.5\ s$	(r)	LS and $T_s = 0.1\ s$

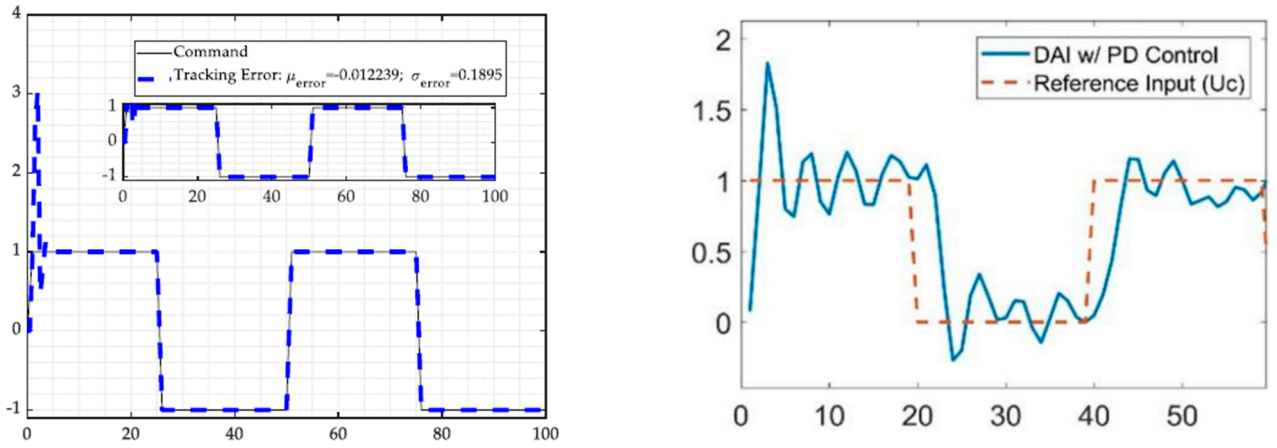
[1] ZOH: zero-order hold; FOH: first-order hold; IIM: impulse-invariant mapping; TA: Tustin approximation; ZPM: zero-pole matching equivalents; LS: least squares.

For each of the discretization methods, sample time lower than 0.1 s results in significant error and deviation from the desired output for the deterministic artificial intelligence method. The plots (a), (d), (g), (j), (m), and (p) in Figure 6 represent the largest permissible sample time value for the deterministic artificial intelligence approach for each discretization method. A sample time larger than the permissible value leads to incorrect results. Based on the plots obtained by running the simulations using Equation (1) and discretizing it using various methods and sample times, the mean of the absolute error between the output and the reference trajectory and the standard deviation of the error are tabulated in the table in Section 4. The lowest mean error for each method is also highlighted.

4. Discussion

The prime contribution of this work is to demonstrate the effects of changing the discretization method and sample time on the control of DC motors used in unmanned underwater vehicles. This work directly uses the continuous transfer function and discretizes it to implement various adaptive control strategies together with learning-based deterministic AI. Figure 7a shows that deterministic artificial intelligence yields lower mean and standard deviation of error in input trajectory tracking when compared with modeling techniques such as indirect self-tuner without process zero cancellation and minimum phase plant presented in [16]. However, the latest literature states that using deterministic artificial intelligence yields an error of 0.224 [19], and the performance of this approach is theorized to be dependent on the efficacy with discretized implementations. From Table 2, it is seen that using the appropriate sample time and discretization method for Equation (1) provides significantly lower errors and thus demonstrates the superiority of using deterministic artificial intelligence. Each discretization method can provide substantially lower error than stated in [18]. The lowest error of 0.0840 is obtained using the first-order hold method and sample time of 0.7 s, a 62.5% reduction from that stated in [19]. It is also observed that as the sample time is reduced, the performance of the deterministic artificial intelligence decreases as well, leading to delays and errors in input trajectory tracking. This result is surprising and not in accordance with conventional logic and intuition. An explanation for this demands research that needs to be carried out in future work.

Recommended future research. *In this work, the sample time is changed in steps of 0.1 s. Reducing this further, a functional relationship could be generated between the sample time and the error statistics for the deterministic artificial intelligence approach. The advantages of doing this would be twofold. Firstly, it might provide even better results for the deterministic artificial intelligence approach. Secondly, comparing the error statistics of the approach with similar statistics from the other methods used in this work would provide quantitative results about the sample time at which the performance of the adaptive approach supersedes the performance of the learning-based approach. Further research should also be carried out to obtain a theoretical and practical explanation for the poor performance of deterministic artificial intelligence with reduced sample times.*



(**a**) (**b**)

Figure 7. (**a**) Results presented in [16] for deterministic artificial intelligence tracking command trajectory. (**b**) Plots for the results obtained in most recent research [18], which shows that the performance of deterministic artificial intelligence is concerned with the efficacy of discretized implementations.

Table 2. Mean and standard deviation of the error for various discretization methods and sample times.[1]

Sample Time (s)	ZOH	FOH	Impulse Invariant Mapping	Tustin Approximation	Zero-Pole Matching Equivalents	Least Squares
0.8	–	–	–	0.1041/0.1646	–	–
0.7	–	**0.0840/0.1430**	–	**0.0894/0.1322**	–	–
0.6	0.1969/0.2683	0.0886/0.1297	–	0.0996/0.1403	–	**0.0912/0.1451**
0.5	**0.1137/0.1612**	0.1083/0.1513	**0.0967/0.1437**	0.1209/0.1658	**0.1219/0.1722**	0.0998/0.1416
0.4	0.1408/0.1897	0.1453/0.1953	0.1179/0.1628	0.1604/0.2141	0.1434/0.1932	0.1329/0.1802
0.3	0.2018/0.2647	0.2148/0.2771	0.1782/0.2331	0.2341/0.3012	0.2037/0.2669	0.1983/0.2571
0.2	0.3405/0.4225	0.3691/0.4512	0.3169/0.3903	0.3981/0.4849	0.3415/0.4238	0.3452/0.4232
0.1	0.8467/0.9812	0.8708/1.0032	0.8045/0.9288	0.9040/1.0405	0.8474/0.9822	0.8413/0.9698

[1] The smallest combination of error mean and standard deviation is highlighted for each method.

Author Contributions: Conceptualization, J.M. and T.S.; methodology, J.M. and T.S.; software, J.M. and T.S..; validation, J.M. and T.S.; formal analysis, J.M.; investigation, J.M.; resources, T.S.; data curation, J.M.; writing—original draft preparation, J.M.; writing—review and editing, J.M. and T.S.; visualization, J.M. and T.S.; supervision, T.S.; project administration, T.S.; funding acquisition, T.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data supporting reported results can be found by contacting the author at jcm483@cornell.edu.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

This appendix provides the entire MATLAB® code that was implemented to obtain the results presented.

```
%% Investigating the effects of discretization for control of DC motors using deterministic
artificial intelligence

clear; close all; clc;

rng(100);       %% Setting up the problem:

% Generate reference input as a square wave
maxtime = 200;
U_ctrl = zeros(1,201);
for i = 1:length(U_ctrl)
      if (mod(floor(i/20),2) == 0)
         U_ctrl(i) = 1;
      else
         U_ctrl(i) = 0;
      end
end
U_ctrl_traj = zeros(1,length(U_ctrl));
check = 1;
next_run = 0;
for i = 1:length(U_ctrl)-1
      if (check)
         U_ctrl_traj(i) = U_ctrl(i);
         delta = U_ctrl(i + 1)-U_ctrl(i);
         i_last = i;
         val_last = U_ctrl(i);
      end
      if (delta ~= 0)
         check = 0;
         if (next_run)
               U_ctrl_traj(i) = val_last + delta/2*(1 + (sin(0.2*pi*(i-i_last)-pi/2)));
         end
         next_run = 1;
         if (U_ctrl_traj(i) == U_ctrl(i) && (i ~= i_last))
               check = 1;
               next_run = 0;
         end
      end
end
U_ctrl = U_ctrl(1:200);

% System modeling
Ts = 0.7;
TF_c = tf(1,[1 1 0]);                                          % continuous transfer
function
TF_d = c2d(TF_c,Ts,'foh');                                     % discrete transfer
function
% Hd = tf([0 0.1065 0.0902],poly([1.1 0.8]),0.5);                % analytic discrete
transfer function
% Hd = tf([0 0.0902 0.06461],[1 -1.213 0.3679],0.5);             % offline discrete
transfer function
Num_coef = [TF_d.Numerator{1,1}(1,1),TF_d.Numerator{1,1}(1,2),TF_d.Numerator{1,1}(1,3)];
Denom_coef =
[TF_d.Denominator{1,1}(1,1),TF_d.Denominator{1,1}(1,2),TF_d.Denominator{1,1}(1,3)];
nzeros = 5;
noise_std = 25;
Noise_distr = 1/noise_std*randn(1,maxtime + nzeros);
```

```
%% Algorithm for Recursive Least Squares (RLS):
% Setup system parameters
b_1 = 0.2; b_0 = 0.1; a_2 = 0; a_1 = 0;
B_m = [0 0.1065 0.0902]; A_m = poly([0.2 + 0.2j 0.2-0.2j]);
a_0 = 0; a_m2 = A_m(3); a_m1 = A_m(2);
time = zeros(1, nzeros); Y_RLS = zeros(1, nzeros); Ym_RLS = zeros(1, nzeros);
U_RLS = ones(1, nzeros); Uc_RLS = [ones(1, nzeros), U_ctrl];
Pmatrix = [100 0 0 0;0 100 0 0;0 0 1 0;0 0 0 1]; THETA_hat_RLS(:,1) = [-a_1 -a_2 b_0 b_1]';
alpha = 0.5; beta = []; gamma = 1.2; lambda = 1.0; Rmatrix = [];

%%%%%%%%%%%%%%%%%%%%%%%%RECURSIVE LEAST
SQUARES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:1:maxtime
    t = i + nzeros; phi_vec = []; time(t) = i;
    Y_RLS(t) = [-Denom_coef(2) -Denom_coef(3) Num_coef(2) Num_coef(3)]*[Y_RLS(t-1)
Y_RLS(t-2) U_RLS(t-1) U_RLS(t-2)]'+...
        Noise_distr(t-1) + Noise_distr(t-2);
    Ym_RLS(t) = [-A_m(2) -A_m(3) B_m(2) B_m(3)]*[Ym_RLS(t-1) Ym_RLS(t-2) Uc_RLS(t-1)
Uc_RLS(t-2)]';
    BETA = (A_m(1) + A_m(2) + A_m(3))/(b_0 + b_1); beta = [beta BETA];

    % Implementation of the RLS method
    phi_vec = [Y_RLS(t-1) Y_RLS(t-2) U_RLS(t-1) U_RLS(t-2)]';
    Kmatrix = Pmatrix*phi_vec*1/(lambda + phi_vec'*Pmatrix*phi_vec);
    Pmatrix = Pmatrix -
Pmatrix*phi_vec*inv(1+phi_vec'*Pmatrix*phi_vec)*phi_vec'*Pmatrix/lambda;
% RLS-EF
    error(i) = Y_RLS(t) - phi_vec'*THETA_hat_RLS(:,i);
    THETA_hat_RLS(:,i + 1) = THETA_hat_RLS(:,i) + Kmatrix*error(i);
    a_2 = -THETA_hat_RLS(2,i + 1); a_1 = -THETA_hat_RLS(1,i + 1);
    b_1 = THETA_hat_RLS(4,i + 1); b_0 = THETA_hat_RLS(3,i + 1);
    Bf(:,i) = [b_0 b_1]'; Af(:,i) = [1 a_1 a_2]';

    % Determine R,S, & T for CONTROLLER
    r_1 = (b_1/b_0) + (b_1^2-a_m1*b_0*b_1 + a_m2*b_0^2)*(-b_1 +
a_0*b_0)/(b_0*(b_1^2-a_1*b_0*b_1 + a_2*b_0^2));
    s_0 = b_1*(a_0*a_m1-a_2-a_m1*a_1 + a_1^2 + a_m2-a_1*a_0)/(b_1^2-a_1*b_0*b_1 +
a_2*b_0^2)...
        +b_0*(a_m1*a_2-a_1*a_2-a_0*a_m2 + a_0*a_2)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2);
    s_1 = b_1*(a_1*a_2-a_m1*a_2 + a_0*a_m2-a_0*a_2)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2) + ...
        b_0*(a_2*a_m2-a_2^2-a_0*a_m2*a_1 + a_0*a_2*a_m1)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2);
    S = [s_0 s_1]; R = [1 r_1]; Rmatrix = [Rmatrix r_1]; T = BETA*[1 a_0];

    % Calculate control signal
    U_RLS(t) = [T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_RLS(t) Uc_RLS(t-1) U_RLS(t-1) Y_RLS(t)
Y_RLS(t-1)]';
    U_RLS(t) = 1.3*[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_RLS(t) Uc_RLS(t-1) U_RLS(t-1) Y_RLS(t)
Y_RLS(t-1)]';
end
%%%%%%%%%%%%%%%%%%%%%%%%END OF RECURSIVE LEAST
SQUARES%%%%%%%%%%%%%%%%%%%%%%%%

%% Algorithm for Autoregressive moving average (ARMA):
% Setup system parameters
b_1 = 0.2; b_0 = 0.01; a_2 = 0; a_1 = 0;
B_m = [0 0.1065 0.0902]; A_m = poly([0.2 + 0.2j 0.2-0.2j]);
```

```
a_m2 = A_m(3); a_m1 = A_m(2); a_0 = 0; n = 8;
time = zeros(1, nzeros); Y_ARMA = zeros(1, nzeros);
U_ARMA = ones(1, nzeros); Uc_ARMA = [ones(1, nzeros), U_ctrl];
THETA_hat_ARMA = zeros(4, maxtime); beta = [];
THETA_hat_ARMA(:,1) = [-a_1 -a_2 b_0 b_1]';
Pmatrix = 10000*eye(n); Pmatrix(1,1) = 1000; Pmatrix(2,2) = 100; Pmatrix(3,3) = 100;
Pmatrix(4,4) = 10000; Pmatrix(5,5) = 1000; Pmatrix(6,6) = 100;
phi_vec = []; Rmatrix = []; lambda = 1;

%%%%%%%%%%%%%%%%%%%%AUTOREGRESSIVE MOVING
AVERAGE%%%%%%%%%%%%%%%%%%%%
for i = 1:1:maxtime
    t = i + nzeros; time(t) = i;
    Y_ARMA(t) = [-Denom_coef(2) -Denom_coef(3) Num_coef(2) Num_coef(3)]*...
        [Y_ARMA(t-1) Y_ARMA(t-2) U_ARMA(t-1) U_ARMA(t-2)]' + ...
        Noise_distr(t-1) + Noise_distr(t-2);          % Generate truth output
    BETA = (A_m(1) + A_m(2) + A_m(3))/(b_0 + b_1); beta = [beta BETA];
    phi_vec = [phi_vec; Y_ARMA(t-1) Y_ARMA(t-2) U_ARMA(t-1) U_ARMA(t-2)];
    if (i > 3)
        THETA_hat_ARMA(:,i + 1) = inv(phi_vec'*phi_vec)*phi_vec'*Y_ARMA(1 + nzeros:t)';
    else
        THETA_hat_ARMA(:,i + 1) = THETA_hat_ARMA(:, i);
    end
    a_2 = -THETA_hat_ARMA(2,i + 1); a_1 = -THETA_hat_ARMA(1,i + 1);
    b_1 = THETA_hat_ARMA(4,i + 1); b_0 = THETA_hat_ARMA(3,i + 1);    % Update A & B
coefficients

    % Save final coefficient values for comparison with real values to obtain epsilon errors
    B_f(:,i) = [b_0 b_1]'; A_f(:,i) = [1 a_1 a_2]';

    % Determine R, S, & T for CONTROLLER
    r_1 = (b_1/b_0) + (b_1^2-a_m1*b_0*b_1 + a_m2*b_0^2)*(-b_1 +
a_0*b_0)/(b_0*(b_1^2-a_1*b_0*b_1 + a_2*b_0^2));
    s_0 = b_1*(a_0*a_m1-a_2-a_m1*a_1 + a_1^2 + a_m2-a_1*a_0)/(b_1^2-a_1*b_0*b_1 +
a_2*b_0^2) + ...
        b_0*(a_m1*a_2-a_1*a_2-a_0*a_m2 + a_0*a_2)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2);
    s_1 = b_1*(a_1*a_2-a_m1*a_2 + a_0*a_m2-a_0*a_2)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2) + ...
        b_0*(a_2*a_m2-a_2^2-a_0*a_m2*a_1 + a_0*a_2*a_m1)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2);
    S = [s_0 s_1]; R = [1 r_1]; Rmatrix = [Rmatrix r_1]; T = BETA*[1 a_0];

    % Calculate control signal
    U_ARMA(t) = [T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ARMA(t) Uc_ARMA(t-1) U_ARMA(t-1)
Y_ARMA(t) Y_ARMA(t-1)]';
    % Arbitrarily increased to duplicate text
    U_ARMA(t) = 1.3*[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ARMA(t) Uc_ARMA(t-1) U_ARMA(t-1)
Y_ARMA(t) Y_ARMA(t-1)]';
end
%%%%%%%%%%%%%%%%%%%%END OF AUTOREGRESSIVE MOVING
AVERAGE%%%%%%%%%%%%%%%%%%%%

%% Algorithm for Extended Least Squares (ELS):
% Setup system parameters
b_1 = 0.2; b_0 = 0.01; a_2 = 0; a_1 = 0;
B_m = [0 0.1065 0.0902]; A_m = poly([0.2 + 0.2j 0.2-0.2j]);
a_m2 = A_m(3); a_m1 = A_m(2); a_0 = 0; n = 8;
time = zeros(1, nzeros); Y_ELS = zeros(1, nzeros); Ym_ELS = zeros(1, nzeros);
```

```
U_ELS = ones(1, nzeros); Uc_ELS= [ones(1, nzeros), U_ctrl];
THETA_hat_ELS(:,1) = [-a_1 -a_2 b_0 b_1]'; beta = [];                % initialize P(t), THETA_hat(t) &
Beta
epsln = [zeros(1, nzeros + maxtime)];
Pmatrix = 10000*eye(n); Pmatrix(1,1) = 1000; Pmatrix(2,2) = 100; Pmatrix(3,3) = 100;
Pmatrix(4,4) = 10000; Pmatrix(5,5) = 1000; Pmatrix(6,6) = 100;
Rmatrix = []; lambda = 1; theta_hat_els = zeros(n, 1);

%%%%%%%%%%%%%%%%%%%%EXTENDED LEAST
SQUARES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:1:maxtime
    t = i + nzeros; time(t) = i; phi_vec = []; k = i + nzeros;
    Y_ELS(t) = [-Denom_coef(2) -Denom_coef(3) Num_coef(2) Num_coef(3)]*...
       [Y_ELS(t-1) Y_ELS(t-2) U_ELS(t-1) U_ELS(t-2)]' + ...
       Noise_distr(t-1) + Noise_distr(t-2);                % generate truth output
    Ym_ELS(t) = [-A_m(2) -A_m(3) B_m(2) B_m(3)]*...
       [Ym_ELS(t-1) Ym_ELS(t-2) Uc_ELS(t-1) Uc_ELS(t-2)]';
    BETA = (A_m(1) + A_m(2) + A_m(3))/(b_0 + b_1); beta = [beta BETA];
    phi_vec = [Y_ELS(t-1) Y_ELS(t-2) U_ELS(t-1) U_ELS(t-2) epsln(t) epsln(t-1) epsln(t-2)
epsln(k-3)]';
    Kmatrix = Pmatrix*phi_vec*1/(1 + phi_vec'*Pmatrix*phi_vec);
    Pmatrix = Pmatrix-Pmatrix*phi_vec*pinv(1 + phi_vec'*Pmatrix*phi_vec)*phi_vec'*Pmatrix;
    error(i) = Y_ELS(k)-phi_vec'*theta_hat_els(:, i);
    theta_hat_els(:, i + 1) = theta_hat_els(:, i) + Kmatrix*error(i);
    epsln(k) = Y_ELS(k) - phi_vec'*theta_hat_els(:, i + 1);    % Posterior Residual formulation
    THETA_hat_ELS(:, i + 1) = theta_hat_els(1:4, i + 1);

    % Update A & B coefficients
    a_1 = -THETA_hat_ELS(1,i + 1); a_2 = -THETA_hat_ELS(2,i + 1);
    b_0 = THETA_hat_ELS(3,i + 1); b_1 = THETA_hat_ELS(4,i + 1);

    % Store final A and B for comparison with real A&B to generate epsilon errors
    A_f(:,i) = [1 a_1 a_2]'; B_f(:,i) = [b_0 b_1]';
    r_1 = (b_1/b_0) + (b_1^2-a_m1*b_0*b_1 + a_m2*b_0^2)*(-b_1 +
a_0*b_0)/(b_0*(b_1^2-a_1*b_0*b_1 + a_2*b_0^2));
    s_0 = b_1*(a_0*a_m1-a_2-a_m1*a_1 + a_1^2 + a_m2-a_1*a_0)/(b_1^2-a_1*b_0*b_1 +
a_2*b_0^2) + ...
        b_0*(a_m1*a_2-a_1*a_2-a_0*a_m2 + a_0*a_2)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2);
    s_1 = b_1*(a_1*a_2-a_m1*a_2 + a_0*a_m2-a_0*a_2)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2) + ...
       b_0*(a_2*a_m2-a_2^2-a_0*a_m2*a_1 + a_0*a_2*a_m1)/(b_1^2-a_1*b_0*b_1 + a_2*b_0^2);
    S = [s_0 s_1]; R = [1 r_1]; Rmatrix = [Rmatrix r_1]; T = BETA*[1 a_0];

    % Calculate control signal
    U_ELS(t) = [T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ELS(t) Uc_ELS(t-1) U_ELS(t-1) Y_ELS(t)
Y_ELS(t-1)]';
    U_ELS(t) = 1.3*[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ELS(t) Uc_ELS(t-1) U_ELS(t-1) Y_ELS(t)
Y_ELS(t-1)]';
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%END OF EXTENDED LEAST
SQUARES%%%%%%%%%%%%%%%%%%%%%%

%% Algorithm for Deterministic Artificial Intelligence (DAI):
% Create command signal
nzeros = 5; time = zeros(1, nzeros);
U_DAI = ones(1,nzeros); Y_DAI = zeros(1, nzeros);                % initialize ouput
vectors
t = 0:200;
hvy_m = [zeros(1, nzeros) U_ctrl_traj];
eb = Y_DAI(1) - hvy_m(1);
```

```
kd = 6.0; kp = 2.0; err = 0;
phid = []; hatvec = zeros(4,1); Rmatrix = []; ustar = [];

%%%%%%%%%%%%%%%%%%%DETERMINISTIC
AI%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Loop through the output data Y(t)
for i = 1:1:maxtime + 1
    t = i + nzeros; time(t) = i;
    d_e = err - eb; u = kp*err + kd*d_e; U_DAI(t-1) = u;
    Y_DAI(t) = [-Denom_coef(2) -Denom_coef(3) Num_coef(2) Num_coef(3)]*...
       [Y_DAI(t-1) Y_DAI(t-2) U_DAI(t-1) U_DAI(t-2)]' + ...
       Noise_distr(t-1) + Noise_distr(t-2);
    phid = [phid; Y_DAI(t) -Y_DAI(t-1) Y_DAI(t-2) -U_DAI(t-2)];
    ustar = [ustar; u]; newest = phid\ustar; hatvec(:,i) = newest;
    eb = err; err = hvy_m(t)-Y_DAI(t);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%END OF DETERMINISTIC
AI%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

THETA_hat_DAI = [hatvec(2,:)./hatvec(1,:); hatvec(3,:)./hatvec(1,:);...
    ones(1,201)./hatvec(1,:); hatvec(4,:)./hatvec(1,:)];

mean_abs_error_DAI = mean(abs(traj_Uc - Y_DAI(1,6:end)))
mean_abs_error_ELS = mean(abs(Uc - Y_ELS(1,6:end)))
mean_abs_error_ARMA = mean(abs(Uc - Y_ARMA(1,6:end)))
mean_abs_error_RLS = mean(abs(Uc - Y_RLS(1,6:end)))
std_error_DAI = std(traj_Uc - Y_DAI(1,6:end))
std_error_ELS = std(Uc - Y_ELS(1,6:end))
std_error_ARMA = std(Uc - Y_ARMA(1,6:end))
std_error_RLS = std(Uc - Y_RLS(1,6:end))

%% Plotting results:
figure(); hold on; grid on;
plot(time(1,1:205), Y_RLS,'g–*'); plot(time(1,1:205), Y_ARMA,'b–o');
plot(time(1,1:205), Y_ELS,'r-.'); plot(time, Y_DAI,'m:');
xlabel('Time step (in sec)'); ylabel('Output (Y)');
plot(time(1:200),Uc,'k-');
axis([0 50,-0.5 2.0]);
% legend('RLS','ARMA','ELS','DAI','Reference Input');
title("Discretization using First Order Hold and Sample Time "+ ts);
PrepFigPresentation(gcf);

function PrepFigPresentation(fignum)

% Prepares a figure for presentations
% Font size: 10
% Font Name: Times
% LineWidth: 2
%
figure(fignum);
fig_children = get(fignum,'children');                    % find all sub-plots

for i = 1:length(fig_children)
    set(fig_children(i),'FontSize',10);
    set(fig_children(i),'FontName','times');
    fig_children_children = get(fig_children(i),'Children');
    set(fig_children_children,'LineWidth',2);
end
end
```

Appendix B

General specifications for DC motors that can be controlled using the approach described in this paper.

Low Voltage DC Motors					
Diameter (mm)	**Input Voltage (V)**	**No Load Speed (rpm)**	**Maximum Efficiency (%)**	**Stall Torque (mNm)**	**Max. Output Power (W)**
15.5	5.0	12,623	50	2.09	0.69
20.4	13.0	25,000	65	20.00	15.00
20.4	2.4	6200	65	8.00	1.30
24.2	2.4	7000	70	26.00	5.00
24.2	1.2	7800	60	12.00	2.50
24.2	24.0	26,000	70	85.00	60.00
24.0	21.0	30,000	60	40.00	32.00
24.4	12.0	8200	62	25.50	5.50
27.5	24.0	7200	55	20.00	4.00
27.5	41.0	18,000	65	38.00	18.00
27.5	18.0	9987	57	39.24	10.20
27.5	39.0	21,000	70	70.00	40.00
27.5	24.0	22,000	70	60.00	35.00
27.5	7.2	17,230	64	114.86	51.24
27.5	36.0	11,000	70	90.00	25.00
27.5	28.0	19,000	75	140.00	70.00
29.0	42.0	6400	64	92.00	15.50
42.3	18.0	20,950	78	1175.03	644.74
48.0	14.4	20,120	66	787.72	415.00
48.0	18.0	20,281	69	656.65	348.79
48.0	18.0	19,600	66	1055.00	542.00
48.0	18.0	22,500	76	1400.00	830.00
High Voltage DC Motors					
Diameter (mm)	**Input Voltage (V)**	**No Load Speed (rpm)**	**Maximum Efficiency (%)**	**Torque @ Max. Efficiency (mNm)**	**Speed @ Maximum Efficiency (rpm)**
35.8	60.0	8600	70	25.00	7400
45.0	230.0	15,600	65	92.00	11,500
52.4	120.0	11,000	64	155.00	8200

References

1. Liu, Z.; Zhuang, X.; Wang, S. Speed Control of a DC Motor using BP Neural Networks. In Proceedings of the 2003 IEEE Conference on Control Applications, Istanbul, Turkey, 25–25 June 2003; pp. 832–835.
2. Mishra, M. Speed Control of DC Motor Using Novel Neural Network Configuration. Bachelor's Thesis, National Institute of Technology, Rourkela, India, 2009.
3. Hernández-Alvarado, R.; García-Valdovinos, L.G.; Salgado-Jiménez, T.; Gómez-Espinosa, A.; Fonseca-Navarro, F. Neural Network-Based Self-Tuning PID Control for Underwater Vehicles. *Sensors* **2016**, *16*, 1429. [CrossRef] [PubMed]
4. Rashwan, A. An Indirect Self-Tuning Speed Controller Design for DC Motor Using A RLS Principle. In Proceedings of the 21st International Middle East Power Systems Conference (MEPCON), Cairo, Egypt, 17–19 December 2019; pp. 633–638.
5. U.S. Naval Forces Southern Command | Navy Deploys Unmanned Submersibles in Argentine Submarine Search. 21 November 2017. Available online: https://www.defense.gov/News/News-Stories/Article/Article/1378119/navy-deploys-unmanned-submersibles-in-argentine-submarine-search/ (accessed on 9 February 2023).
6. Rees, C. Maxon Launches High Torque DC Brushless Motors. 5 May 2015. Available online: https://www.unmannedsystemstechnology.com/2015/05/maxon-launches-high-torque-dc-brushless-motors/ (accessed on 19 December 2022).

7. Department of Defense Photographs and Imagery, Unless Otherwise Noted, Are in the Public Domain. Available online: https://www.defense.gov/Help-Center/Article/Article/2762906/use-of-department-of-defense-imagery/#:~:text=Department%20of%20Defense%20photographs%20and,use%2C%20subject%20to%20specific%20guidelines (accessed on 9 February 2023).
8. Available online: https://www.maxongroup.com/maxon/view/content/underwater-drive-systems (accessed on 10 February 2023).
9. Slotine, J.; Benedetto, M. Hamiltonian adaptive control on spacecraft. *IEEE Trans. Autom. Control* **1990**, *35*, 848–852. [CrossRef]
10. Slotine, J.; Weiping, L. *Applied Nonlinear Control*; Prentice Hall: Englewood Cliffs, NJ, USA, 1991.
11. Fossen, T. Comments on "Hamiltonian Adaptive Control of Spacecraft". *IEEE Trans. Autom. Control* **1993**, *38*, 671–672. [CrossRef]
12. Åström, K.; Wittenmark, B. On the Control of Constant but Unknown Systems. In Proceedings of the 5th IFAC World Congress, Paris, France, 12–17 June 1972.
13. Åström, K.; Wittenmark, B. On self-tuning regulators. *Automatica* **1973**, *9*, 185–199. [CrossRef]
14. Åström, K.; Wittenmark, B. *Adaptive Control*; Addison-Wesley: Boston, FL, USA, 1995; pp. 90–135.
15. Sands, T.; Kim, J.; Agrawal, B. Improved Hamiltonian Adaptive Control of spacecraft. In Proceedings of the 2009 IEEE Aerospace conference, Big Sky, MT, USA, 7–14 March 2009.
16. Sheng, M.; Alvi, M.; Lorenz, R. GMR-based Integrated Current Sensing in SiC Power Modules with Phase Shift Error Reduction. *IEEE J. Emerg. Sel. Top. Power Electron.* **2022**, *10*, 3477–3487. [CrossRef]
17. Sands, T. Development of Deterministic Artificial Intelligence for Unmanned Underwater Vehicles (UUV). *J. Mar. Sci. Eng.* **2020**, *8*, 578. [CrossRef]
18. Sands, T. Control of DC Motors to Guide Unmanned Underwater Vehicles. *Appl. Sci.* **2021**, *11*, 2144. [CrossRef]
19. Shah, R.; Sands, T. Comparing Methods of DC Motor Control for UUVs. *Appl. Sci.* **2021**, *11*, 4972. [CrossRef]
20. Mareels, I.; Anderson, B.; Bitmead, R.; Bodson, M.; Sastry, S. Revisiting the Mit Rule for Adaptive Control. *IFAC Proc. Vol.* **1987**, *20*, 161–166. [CrossRef]
21. Sprott, D. Gauss's contribution to statistics. *Hist. Math.* **1978**, *5*, 183–203. [CrossRef]
22. Sands, T.; Kim, J.; Agrawal, B. Spacecraft fine tracking pointing using adaptive control. In Proceedings of the 58th International Astronautical Congress, Hyderabad, India, 24–28 September 2007; International Astronautical Federation: Paris, France, 2007.
23. Smeresky, B.; Rizzo, A.; Sands, T. Optimal Learning and Self-Awareness versus PDI. *Algorithms* **2020**, *13*, 23. [CrossRef]
24. Zhai, H.; Sands, T. Controlling Chaos in Van Der Pol Dynamics Using Signal-Encoded Deep Learning. *Mathematics* **2022**, *10*, 453. [CrossRef]
25. Zhai, H.; Sands, T. Comparison of Deep Learning and Deterministic Algorithms for Control Modeling. *Sensors* **2022**, *22*, 6362. [CrossRef] [PubMed]
26. Åström, K.; Apkarian, J.; Lacheray, H. Quanser Engineering Trainer (QET) Series: USB QICii Laboratory Workbook, DC Motor Control Trainer (DCMCT) Student Workbook. Available online: http://class.ece.iastate.edu/ee476/motion/Main_manual.pdf (accessed on 13 February 2023).